RADC-TR-70-90
Final Technical Report
June 1970

FORMATTED FILE ORGANIZATION TECHNIQUES

IBM Research Laboratory

Rome Air Development Center
Air Force Systems Command
Griffiss Air Force Base, New York

FORMATTED FILE ORGANIZATION TECHNIQUES

Michael E. Senko, et al

IBM Research Laboratory

FOREWORD

This Final Report was submitted by IBM Research Laboraotry, Information Sciences Department, San Jose, California, under Contract F30602-69-C-0097, Project 4594, with Rome Air Development Center, Griffiss Air Force Base, New York. RADC Project Engineer was Miss Patricia Langendorf, EMIDD.

This report has been reviewed by the Information Office, EMLS, and is releasable to the Clearinghouse for Federal Scientific and Technical Information.

This report has been reviewed and is approved.

Approved: Patricia M. Langendorf
PATRICIA M. LANGENDORF
Project Engineer

Approved: [illegible]
HOWARD DAVIS
Acting Chief
Intel and Recon Division

FOR THE COMMANDER: [illegible]
IRVING J. GABELMAN
Chief, Plans Office

ABSTRACT

This Final Contract Report contains papers presenting several useful steps toward the creation of a more scientific discipline of formatted file design.

In particular, there are papers on:

(1) The first extensive, fundamentally oriented, comparison of key-to-address transforms utilizing existing formatted files.

(2) Formal, mathematical descriptions of formatted file systems that are used to provide concepts and means to deal with:

- (a) the selection of indexes;
- (b) direct retrieval on the basis of multiple attributes, and
- (c) questions of storage and response time efficiency.

(3) The calibration of the FOREM I Formatted File Organization Simulation Model.

(4) A new, more powerful 9000 FORTRAN statement model (FOREM II) for simulating the effects of complex file organizations, and machine configurations on efficiency and response times in a formatted file query and update environment.

CONTENTS

INTRODUCTION 1

SECTION I - KEY-TO-ADDRESS TRANSFORM TECHNIQUES: A FUNDAMENTAL PERFORMANCE STUDY ON LARGE EXISTING FORMATTED FILES I-1

SECTION II - A QUANTITATIVE APPROACH TO THE SELECTION OF SECONDARY INDEXES II-1

SECTION III - SOME RESULTS ON STORAGE SPACE REQUIREMENT AND RETRIEVAL TIME IN FORMATTED FILE ORGANIZATION III-1

SECTION IV - ORGANIZATION OF RECORDS WITH UNEQUAL MULTIPLE-VALUED ATTRIBUTES AND COMBINATORIAL QUERIES OF ORDER 2 IV-1

SECTION V - CALIBRATION OF THE FILE ORGANIZATION EVALUATION MODEL (FOREM I) V-1

SECTION VI - PHASE II - A DATA MANAGEMENT SYSTEM MODEL VI-1

SECTION VII - THE PHASE II DATA MANAGEMENT SIMULATION SYSTEM (Level 2) -- USER GUIDE -- VII-1

SECTION VIII - GLOSSARY VIII-1

INTRODUCTION

The field of Formatted File Organization has become increasingly important with the advent of large random access peripheral files and requirements for real-time retrieval and maintenance. It is the purpose of this contract to provide new techniques, tools and information which will lead to a fundamental design discipline for operational files. This design discipline will be solidly based on studies of actual computer hardware systems, software systems, and associated user files.

The original IBM Research Division-IBM Federal Systems Division work in the area was recorded in the Final Report for Rome Air Development Center Contract No. AF 30(602)-4088. This previous report presented for the first time:

(1) Detailed comprehensive surveys of the processing and content of four large intelligence files in unclassified, implementation-independent form;
(2) New techniques for the organization of formatted files for direct multiple attribute retrieval;
(3) A file organization evaluation model (FOREM I) which accepts the survey material in all its detail with respect to file content and transactions (complex queries and updates) and allows the user to evaluate the effects of different file organizations on system efficiency and response time.

This above effort provided a basis for dealing not just with abstract theories, but also with existing files and hardware-software systems. As the result of a further contract No. AF 30602-69-C-0100, a prototype file design handbook was created using information obtained from thousands of actual computer system and simulation model (primarily FOREM I) runs. This handbook represents a first effort to create design guidelines based on extensive empirical data.

The present Final Contract Report represents another step forward in the creation of empirical information, techniques, and practical tools for file design. Here again, the philosophy is that empirical and abstract contributions are necessary for the eventual creation of a science of file design, but that the abstract contributions must be solidly connected with practical understanding of actual systems. The work to be reported covers three areas.

The initial area continues the study of actual files. It is represented by a paper on a fundamentally oriented, comparative study of key-to-address transforms using actual key sets. This paper makes a significant departure from existing literature on key-to-address transforms; it adds no new transform proposal to the many existing, unevaluated, uncompared transforms; instead, on the basis of actual runs, it presents a number of useful facts and guidelines for selecting a transform appropriate to the user's key set. In its conclusions section, it goes further to propose and discuss more fundamental techniques for selecting transforms on the basis of defined characterizations of both transforms and key sets.

The second area continues work on creating a fundamental basis for file design. In this area, two papers present formal, mathematical descriptions of certain aspects of formatted file organizations. These descriptions are then used to provide means and concepts for dealing with the selection of indexes and the questions of storage and response time efficiency. A third paper extends the power of prior work on direct, multiattribute retrieval.

The final area is concerned with the calibration of the FOREM I model and the creation and description of a new, significantly more powerful program for modeling formatted file organizations (FOREM II). This model, which is a 9000 FORTRAN statement program, provides expanded capability in almost all areas over FOREM I; the most important area, however, is the ability to deal with simultaneous operations within a single program, with much more complex machine configurations, and with a wider variety of query formulations.

The model is described in a final paper and by an included copy of the user's guide documentation.

In summary, this Final Contract Report contains papers presenting several useful steps toward the creation of a more scientific discipline of formatted file design.

SECTION I

KEY-TO-ADDRESS TRANSFORM TECHNIQUES: A FUNDAMENTAL PERFORMANCE STUDY ON LARGE EXISTING FORMATTED FILES

V. Y. Lum
P. S. T. Yuen
M. Dodd

KEY-TO-ADDRESS TRANSFORM TECHNIQUES: A FUNDAMENTAL PERFORMANCE STUDY ON LARGE EXISTING FORMATTED FILES*

by

V. Y. Lum
P. S. T. Yuen
M. Dodd

Information Sciences Department
IBM Research Laboratory
San Jose, California

ABSTRACT: This paper presents the results of a study of eight different key-to-address transformation methods applied to a set of existing files. As each method is applied to a particular file, load factor and bucket size are varied over a wide range. In addition, appropriate variables pertinent only to a specific method also take on different values. The performance of each method is summarized in terms of the number of accesses required to get to a record and the number of overflow records created by a transformation. Peculiarities of each method are discussed. Practical guidelines obtained from the results are stated. Finally, a proposal for further quantitative fundamental study is outlined in the conclusion.

INTRODUCTION

The direct access method normally provides the most rapid means of accessing a single record of a formatted file. In cases where there is one record or nearly one record per possible key value, access requires only a multiplication of the key by the record length to obtain the address of the desired record. The record can then be obtained by only one access to the peripheral file. When there is less than one record for every two or more possible key values, then the direct multiplication transform will leave a considerable amount of empty space for key values that are unused. To reduce the amount of waste space, numerous workers have proposed a means for mapping large key spaces into smaller address spaces. The main problem is that none of the practical instances of these key-to-address transforms can guarantee to produce a uniform mapping of keys to addresses for any arbitrary distribution of key values. Given this situation, one needs guidance on the selection of a technique that will produce the most nearly uniform distribution for his practical situation.

Unfortunately, up to this time, workers in the field have devoted their efforts toward inventing new transforms rather than toward creating guidelines by comparing existing ones in practical situations so that their relative performance can be characterized. The only comparative evaluation known to the authors appears in Buchholz.[1] He presents an excellent discussion of various aspects of key-to-address transformation, but his experimental results are minimal. In this paper, we undertake to provide a major experimental, comparative evaluation of several transform techniques and have obtained several pragmatic user guidelines for the selection of an appropriate practical transform.

Based on this information, we also discuss in the conclusions section a possibly more quantitative, fundamental approach to transform selection. In particular, we seek to define two sets of characterization functions which may be applied to key sets and to transforms. If the characterization functions are valid, then we should be able to make quantitative comparisons between the characterization function values for a particular key set and the characterization function values for proposed transforms to decide which transform is likely to perform best.

EXPERIMENTAL ENVIRONMENT

There are many factors affecting the performance of key-to-address transform techniques. This section contains a discussion of the dominant parameters considered in the present experiments. The topics presented include:

(1) the key set sample
(2) the transformation method
(3) the variable parameters
(4) the method of handling overflows or clashes.

The Sample Key Sets

Characteristics of the eight files used in this experiment are shown in Table I. The eight key sets contain files of different sizes with a variety of key types. Some keys are long, some are short, some alphanumeric, and some numeric. In addition, some files have keys densely distributed in the key space, and some sparsely distributed. Because of its diversity, it is believed that this sample is representative of the general range of files. The selected transformation methods will be applied to each of these files.

Variables

As mentioned in many articles,[1,2,5] the two dominant variables affecting performance are loading factor and bucket size. The former is the ratio of the number of records to the number of record slots. (A slot is a unit of storage space that can hold one record.) The latter is the number of records that can be accommodated in an image under a transformation. A decrease in the loading factor reduces the probability that many records will be mapped to the same location and an increase in the bucket size increases the capacity of each image. Both will tend to reduce the number of overflow records. In this experiment, the loading factor is varied from 0.5 to 0.95 at intervals of 0.05. For each choice of the loading factor, the bucket size takes on the values of 1, 2, 5, 10, 20 and 50.

	Number of Records	Type*	Key Length (in no. of symbols)**
County State Code	3072	N	5
Personnel	2241	N	6
Personnel Location	930	AN	7
Applicants	762	N	6
Customer Code	24050	N	6
Product Code	33575	N	6
Library	4909	A	12
Random Numbers	500	A	10

*A = alpha, N = numeric, AN = alphanumeric

**A symbol can be a digit, letter, etc.

TABLE I

There are other variables pertinent only to a specific transformation. They will be described in the discussion of the appropriate methods.

Transformation Methods

Six different techniques, producing eight different transformation methods have been studied. Each method transforms the keys into addresses with bucket size equal to 1. In the case of a larger bucket size, the bucket addresses are determined by a modulo B operation, where B is the number of buckets available. An alternative is to map the key directly into bucket addresses in one process. However, it was found from the tests made that the alternative showed no significant difference from the first approach. Because of the way some of the methods operate, the modulo operation cannot be eliminated from these methods. Consequently, it was decided that the first approach would be used throughout the experiment.

(i) Division - Undoubtedly, the best known and most frequently used technique is division of the key by a positive integer, particularly a prime number. In this method, the remainder obtained from the division becomes the address for the key. The divisor, q, is usually chosen to be approximately equal to the number of available addresses, M. Buchholz[1] suggested a refinement that q be the largest prime number smaller than M. The utility of his suggestion is not so obvious. Given that a key distribution contains clusters of various sizes at random with gaps of different lengths also at random, it may be that the choice of any q equal to or near M will perform just as well. One set of experiments was performed to check the truth of this conjecture.

(ii) Digit Analysis - In this method, the distribution of values of the keys in each position or digit (where digit is not necessarily a decimal digit) is determined. Those positions having the most skewed distributions will be deleted from the key until the number of remaining digits is equal to the desired address length, which is the number of digits in the highest slot number. The criteria adopted to find the digits to be used as addresses, based on the measure of uniformity in the distribution of values in each digit, is to keep those positions having no abnormally high peaks or valleys and those having small standard deviations. In a given file, the same digits must be dropped from all keys.

Digit analysis is the only method investigated that exploits key distribution and it is only a partial exploitation. There do exist "perfect" transformations which exploit knowledge of the key distributions to produce perfectly uniform distributions of addresses. These transformations generally require extremely extensive manipulations of the key sets and are not practical for data bases that receive even a single new update record. Our study is therefore confined to the practical "non-perfect" transformations that cannot guarantee perfectly uniform distributions for arbitrary key sets.

(iii) Mid-Square - A key is multiplied by itself and its address is obtained by truncating digits at both ends of the product until the number of digits left is equal to the desired address length. As in the digit analysis method, the same positions must be kept from all products.

(iv) Folding - A key is partitioned into a number of parts each of which, except the last, has the same length as the address length. (There are methods which partition a key into shorter parts. These methods have not been investigated here because it is believed that their characteristics are about the same as the ones studied.) Two methods have been investigated. One folds the key at the boundary of the parts as if folding paper. Digits falling into the same position will be added together. The other method is to shift over the sections so that the lower ends of the sections align before carrying out the addition These two methods will be referred to as fold-boundary and fold-shifting, respectively. In either case, decimal addition is used. Figure 1 below illustrates the positional manipulation of the two methods.

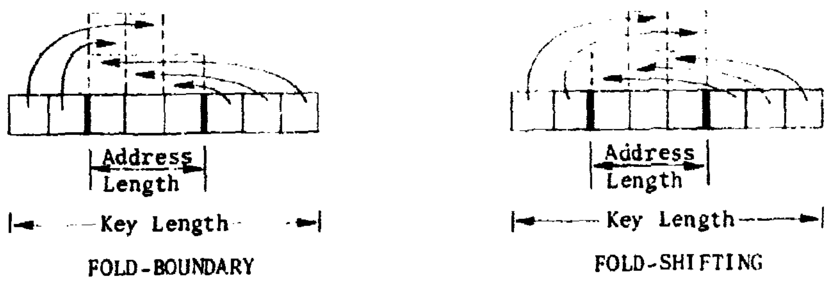


FIGURE 1

(v) Lin's Method - In this method, a key is expressed in radix p and the result taken modulo q^m where p and q are relatively prime and m is a positive integer. Given a key, it is first written as a simple binary bit string. These bits are then grouped to form p-nary digits. The result is expressed as a decimal number which, taken modulo q^m, gives the address. To simplify the selection of p, q, and w, it was decided to have $p = q + 1$, and p and m are so chosen that q^m approximates the number of addresses available.

Let us illustrate the process with an example. Suppose that the key is 975. Encoding each of the digits with 4 bits (the smallest number of bits required to represent a decimal digit), we have a binary string of 100101110101. Now, if the number of addresses available is 48, the choice of $p = 8$, $q = 7$, and $m = 2$ conforms to the rule defined. Grouping three bits together, we have $4565)_8 = 2421)_{10}$. The address, then, is given as 20 by taking 2421 modulo 49. (Note that if p had been 10, then the address would have been obtained simply by taking the key modulo q^m, i.e., same as the division method.) Note that there are different ways to express a key in a binary vector, e.g., BCD or binary. Our investigation showed that the results do not vary significantly for these cases.

The details in this mapping may not be exactly as proposed by Lin,[2] but the principle remains the same.

(vi) Algebraic Coding - Each digit of a key is considered to be a polynomial coefficient. The polynomial so obtained is divided by another polynomial $g(x)$ which is invariant for all the keys in a set. The coefficients of the remainder polynomial form the address.

This method, based on the theory of error correcting codes,[3,4,12] assures that if $g(x)$ is chosen in such a manner that all polynomials containing $g(x)$ as a factor have a minimum weight or distance (Hamming distance) of d, then no two keys differing by d or less positions can be mapped to the same address. Application of this theory requires the coefficients of $g(x)$ and $k(x)$, the division and the polynomial from the key, respectively, to be elements of a Galois field[12] of q elements, $GF(q)$, where q is a power of a prime. Thus, a decimal key must have as its components elements in GF(q) with $q > 10$. Alternatively, one may expand the key into a vector or a string of elements with the value of each element smaller than 10. Two Galois fields, GF(2) and GF(16) have been chosen more or less arbitrarily for this study.

The selection of $g(x)$ was done in two ways: (1) selecting $g(x)$ on the basis of distance, and (2) selecting $g(x)$ randomly with the restriction that neither the highest degree coefficient nor the constant term is zero. The degree of $g(x)$, of course, is determined from the size of the storage available so that the remainder can cover the range of addresses. More precise and detailed discussions on this method can be found in Peterson,[12] Schay,[3] and Hanan.[4]

In addition to mapping a set of keys into addresses using one of the methods discussed, it is also possible to create a transformation with a combination of two or more methods. For example, one may first multiply a key by itself and the product is then folded to form an address. Here, mid-square and folding are used in conjunction. Lin's method is essentially a combination of two basic methods: radix transformation and division. (Actually, nearly all transformation methods require the application of the division method to find the bucket addresses.) In this experiment, combining methods will not be studied because it is believed that the characteristics of the individual methods determine the characteristics of a combination.

Alphanumeric Keys

Since some of the key sets are alphabetic or alphanumeric and since nearly all the transformation methods operate only on numerical values, it becomes necessary to encode the alphabetic or alphanumeric keys as numeric keys. Several different encoding schemes have been tried. No significant variation in performance was discovered provided the encoding schemes preserve distinctness of the symbols. The scheme finally selected and used throughout the experiment is to encode the letters a, b, c, ..., z into decimal numbers 11, 12, ..., 36. Numerical digits remain unchanged. It is understood that key length refers to the number of digits after encoding.

Overflow Storage

In general, the available memory or address space is divided into small sections called buckets. Every record will be mapped into one of these buckets. Since non-perfect transformation techniques may map an excessive number of records into the same bucket, methods must be devised to handle overflow records which cannot be accommodated by their home addresses.

The two basic techniques commonly adopted to accommodate overflow records are:
(1) storing them in vacancies in another bucket in the prime area, and
(2) storing them in a separate or independent overflow area. Many variations are possible in each basic technique. For example, one version of the first technique is the search for vacancies successively starting from a record's home bucket. The process continues until an accommodation is found. (Storage space is considered to be circular and the amount of storage space must be large enough to hold all the records.) This technique, proposed by Peterson,[5] is usually called the open addressing or consecutive spill method. A variation of this method is to search for space, whenever a record's home bucket is filled, by skipping a number of buckets as defined by a selected rule.[9,15,16] When this skipping technique is used, one should select a rule such that the entire storage space can be searched when necessary.

A basic version of the separate overflow method is chaining. Here, the location of the first overflow record from each bucket is listed in the record's home bucket. Pointers are stored in each successive overflow record in the chain to indicate the address of the next record. A variation of the separate overflow technique is to provide small areas, each of which can only be used to store overflow records from a particular section of the prime area. Overflow records that cannot be accommodated here go to a larger, independent area available to all buckets.

Our study will be limited to the basic techniques of open addressing and of chaining in separate overflow areas.

Performance Measure

In order to compare the performance of various transformation techniques, a standard of measurement must be established. After an investigation of various approaches, the average number of accesses per record and the number of overflow records were found to be the most appropriate performance indicators.

In open addressing, the number of accesses for a given record is equal to $S + 1$, where S is the number of buckets away from that record's home bucket. In chaining, each record located in its home bucket is said to require one access. A given overflow record is said to need $T + 1$ accesses, where T is its chain position. For each transformation, the average number of accesses per record and the percentage of overflow records for each key set have been calculated. Further averages over the entire eight key sets were then computed.

EMPIRICAL RESULTS AND DISCUSSION

Tables II to XXI present the results of the study. Summarized in the tables are the average accesses per record for the two different overflow storage techniques, the average percentage of overflows for each transformation, and the standard errors. It can be seen from these tables that the division technique gives the best overall performance and that the mid-square technique is a close second. In fact, the mid-square method has the lowest number of accesses per record for open addressing and loading factors below 0.75. The mid-square method also provides the most consistent performance as evidenced by the small standard error for the various key distributions. Among the other methods, the algebraic technique is good when chaining of overflow records is used. Lin's method is consistently poor; folding and digit analyses are erratic.

All the transformation technques display the same performance trend; namely, the number of accesses per record and the percentage of overflow records increases with higher loading factor and decreases with larger bucket size. The changes are gradual when chaining overflow is used, but they are very drastic for open addressing with small bucket size, i.e., 1, or 2. Indeed, open addressing performance for small bucket sizes is so erratic that, even with a loading factor of only 0.5, there are cases which require more than 800 accesses on the average to retrieve a record. From the results obtained, one can obviously conclude that open addressing should not be used for small bucket sizes.

The results for open addressing with small bucket sizes are much worse than those obtained in Peterson's experiment.[5] The discrepancy undoubtedly is caused by Peterson's idealized assumption that a Poisson distribution of addresses would result from the transformations.

When bucket size becomes larger, open addressing improves rapidly. At 20 or 50, it outperforms chaining in general. However, because of the small number of overflow records at these bucket sizes, the difference is very slight. Hence, the use of either technique will be equally satisfactory.

When tabulating the results of the transformation methods on the key sets before averages are taken, it was found that no mapping method is consistently the best. For example, the two methods using the folding technique are excellent in some of the files but because of one or two poor results (not necessarily the same ones in the two different kinds of folding), degradation in performance occurs after averaging. The same phenomenon occurs for all transformation methods. If, before averaging, one or two of the poor results are removed from the data of each transformation, then nearly all the techniques will show about the same performance.

Every method of transformation has its idiosyncrasies. Let us briefly discuss each.

Consider first the method of simple division. As mentioned before, the keys are believed to be distributed in clusters of various sizes separated by gaps of different lengths. If this assumption is correct, the choice of a divisor. becomes immaterial. An experiment was designed to shed some light on the subject. Prime, odd but not prime, and even numbers have been chosen as divisors.

In each of the three categories tested, an abrupt change in performance sometimes occurs for a small variation in loading factor and/or bucket size. The frequency of occurrence of this behavior is less than 2% of all cases tested for the prime divisors, about 2% for the odd divisors and about 10% for the even divisors. The abruptness is much more pronounced in the case of even divisors. However, most of the ten percent occurs in the results of two of the key sets. Detailed investigation revealed that each of the key sets giving poor results with even divisors has a preponderance of odd numbers. Since evenness and oddness are preserved after division by an even number, a skew distribution of addresses exists after transformation and poor performance therefore arises. In general, if a large number of keys are equal modulo d and if D is a multiple of d, then the use of D as a divisor will result in poor performance. This is the basic argument, as suggested by Buchholz, that the largest prime number close to but less than the size of the address space should be selected as the divisor. However, it appears that most non-prime numbers are valid candidates for divisors since inferior results are relatively rare, and since they often outperform the prime numbers. It is advisable, nevertheless, to choose divisors which do not contain small prime numbers as factors. This, of course, eliminates even numbers.

In Lin's method, the choice of a slightly different p can change the results drastically. The reason for this is not known. Lin[2] showed in his experiment that his technique produced addresses close to a Poisson distribution. Our data also tend to substantiate Lin's claim. However, as Buchholz believes, perfect randomization (Poisson distribution of addresses) is not a desirable goal. Our experimental results confirm his belief. All transformations (none of which produce perfect randomization) give better performance than true randomization.[11]

Transformation by digit analysis is not recommended. Even with the additional overhead imposed by the analysis, the results were not satisfactory. Truncation by observation may eliminate the analysis but it is also not very reliable.

The mid-square transformation technique, as mentioned before, can be applied with some confidence that fairly good results will be obtained. Although it is not apparent in our tabulated results, this technique can also produce unexpectedly poor performances. For long keys and short addresses, and if the middle digits of the keys vary little, a large number of distinct keys will all be mapped to the same bucket.

It is easiest to compute an address by the method of folding keys when the key length is long and the desired address fits into one computer word. The operations, shifting and adding, are much easier to carry out than the operations associated with other techniques. For key length nearly the same as address length, this method behaves like the division method.

By far the most complicated method is the algebraic coding technique. Here even the choice of a generating polynomial to guarantee a minimum distance in the code is not an easy task. Experiments with codes of various minimum distances in GF(2) have been applied to test the claims of advantages given by the proponents of this method.[3,4] They believed that large minimum distances of a code assure good performance. The data obtained do not substantiate their assertion since there does not seem to exist any correlation between performance and the distance of a code. Neither larger nor smaller distances produce uniformly better results. Codes chosen at random consistently perform equally well. The choice of the Galois field also does not seem to be important. As shown in the tables, the performance figures are nearly the same for both fields used in the experiment.

CONCLUSIONS

Pragmatic Choice of Transforms

Faced with an arbitrary key set, the selection of a transformation technique is obvious; the division method is preferred. While other techniques may sometimes perform better, one also risks obtaining inferior results more often. In the division method, the choice of a divisor is not necessarily limited to prime numbers. Selecting a number which does not contain any prime factor below, say, 20 is probably sufficient to assure good performance.

Overflow Handling

The overflow handling technique to be used depends on bucket size. If the bucket size is less than 10 records, open addressing should not be employed. For larger sizes, this technique can be applied to save storage space and yet maintain good performance. Chained overflow handling, however, generally gives a much more predictable result because overflow records do not affect prime storage space. However, in determining which overflow handling technique is superior, one must take into account the characteristics of the storage device and the operational system. For example, if disks are used, arm motion must be analyzed with the number of accesses given in the experiment. If chained overflow requires a large number of arm movements, then it may become impractical. On the other hand, if the system misses rotations when accessing successive buckets, open addressing may become just as expensive.

Static and Dynamic Data Sets

Although the study here has been limited to static key sets, it is believed that the data obtained are applicable to dynamic situations where keys can be deleted or added. From the results of Olson,[11] it can be seen that the difference between a dynamic situation and its static analog (initial loading in Olson) is relatively very small when compared to the deviations produced by the transformations themselves. Consequently, the results here can still be used as a guide in both situations.

Characterization Functions for Transformations and Key Sets

The previous discussions are pragmatic statements based on the results of our study. This study has led us to a conclusion that it is desirable to create a more quantitative fundamental approach.

As mentioned earlier, a comparison of the data obtained in this study and that based on the Poisson distribution of addresses indicates that the idea of finding a transformation technique that will "randomize" a key set is a misconstrued objective. This misconception is often the result of the belief that a "randomizing" transformation will map the keys into evenly distributed addresses. Actually, an ideal transformation method must map all keys in a file to distinct addresses. Uniformity in the distribution of addresses is not synonymous with the mapping of a key into addresses with equal probability. Consequently, as believed by Buchholz[1] and substantiated by the results in this paper, an efficient transformation method should preserve whatever uniformity exists in the keys.

Based on this information, we would like to find a set of characterization functions for the transformations. We would like to classify these transformation methods with respect to their capabilities in preserving local uniformities in the key set or randomizing the keys into addresses. Since the distribution of a key set also plays an important role in the performance of a transformation, we would also want to define a set of characterization functions for the key sets. If the characterization functions are meaningfully selected, we should be able to determine which transformation is likely to perform well on a given set of keys.

Let us consider the techniques used by the transformation methods. Generally speaking, all transformation methods may be characterized as either distributive or randomizing according to the manner in which the addresses are generated from the keys. A distributive transformation preserves the order of the keys in the resulting addresses to a large extent; a randomizing transformation destroys the order completely.

Let us define more precisely the two terms, distributive and randomizing. Let $k_0, k_1, \ldots, k_{n-1}$ be n numerically consecutive keys. Let $0, 1, 2, \ldots, n-1$ be the range of the mapping, consisting of the addresses of the available slots. n is an arbitrary integer and is fairly large; for purposes of standardization, we choose 1000. Since the slots have distinct addresses, these two terms will be used interchangeably. A transformation T will map each key to one of the addresses. Let us suppose that the keys k_i, $i = 0, 1, \ldots, n-1$ are transformed as given by $s_i = T(k_i) \bmod n$. Note that the images of the mapping are not necessarily distinct and that s_{i+1} does not necessarily follow s_i in the sequence of addresses. A set of $j + 1$ images, $s_i, s_{i+1}, \ldots, s_{i+j}$, from the set of keys $k_i < k_{i+1} < \ldots < k_{i+j}$ is said to be in order if and only if there exists an integer b such that $s_i' < s_{i+1}' < \ldots < s_{i+j}'$ or $s_i' > s_{i+1}' > \ldots > s_{i+j}'$ where $s_\ell' \equiv b + s_\ell \bmod n$. The order is said to be destroyed otherwise. Essentially, the above statement specifies that the addresses are circular and that the set of images are in order if these images can be arranged in one of the two ways just given by cyclic shifts. Let us define the order length of a transformation T to be the integer m given as follows: Let T map the keys $k_0, k_1, \ldots k_r$

into the addresses one at a time, starting from k_0 and carrying on sequentially. Let this process be repeated many times starting from different keys. On the average let the m^{th} key be the first key that is mapped into an address resulting in a destruction of order. Then T is said to have an order length equal to m. T is said to be perfectly distributive if m is equal to n + 1. It is said to be distributive if m is large and randomizing if m is small.

Let us define a second parameter, collision length. Again, let T map the keys into addresses in the same manner as above. On the average let the c^{th} key be the first key mapped into an address such that the addresses of the c keys just obtained are no longer all distinct. (Here the order of the key set is not necessarily preserved.) c is said to be the collision length of T. By definition, a perfectly distributive transformation has a collision length equal to n + 1. A randomizing transformation will give a smaller collision length. However, transformations having small order lengths do not necessarily possess small collision lengths. Conventionally, a measure of the efficiency of a transformation method is the uniformity of the distribution of the addresses. If a file is accessed randomly all the time, this measure, based on uniformity, is satisfactory. However, sometimes sequential retrieval of the keys may be required. Since mass storage devices are usually not truly random, preserving the order of the keys becomes advantageous. For sequential retrieval, a perfect distributive method which preserves order as well as giving a uniform distribution of addresses intuitively seems desirable. On the other hand, if only random accessing capability is concerned, the parameter collision length may be a satisfactory performance indicator of a transformation method. Of course, due to the arbitrary distribution of a key set, a closer to perfect distributive method does not necessarily give better performance for a particular file. It can only be expected to have good results on the average.

Of the methods studied in this paper, the division method is an example of a perfect distribution. Lin's method is much closer to randomization. Categorization of the other methods is not so obvious because their characteristics depend on parameters such as key length, address length, and the kind of operations used. For example, the mid-square method probably has high order and collision lengths except where keys have a string of zeros giving all zeros in the address portion of the product.

In addition to the categorization of the transformation methods, one should also consider the classification of the files by their statistical properties. As mentioned before, it is quite possible that a particular transformation technique works exceptionally well for a certain kind of key distribution and poorly on others. However, before making an investigation in this direction, we should first consider what statistical characteristics are most appropriate.

One approach to the classification problem is to find the underlying distributions from which the keys are obtained. However, because of the discrete nature of the situation and the possible arbitrary selection in selecting a key, one will frequently have difficulty in identifying the underlying distributions. A much easier and still meaningful approach is to find the distribution of the cluster lengths and the gap lengths between clusters in a key set. (A cluster here is a set of numerically consecutive keys separated at both ends from other keys.) Perhaps the means and variances of the cluster lengths and gap lengths will be sufficient to classify a set of keys for our purpose. In addition, the density of the key set, or the ratio of the number of keys in a file to the number of keys possible in the key space, also plays an important role and should be taken into account.

Let us discuss briefly the importance of these parameters. Let m_c, m_g, v_c and v_g be the means and variances of the cluster lengths and gap lengths respectively with the subscripts c and g denoting cluster and gap. Let d be the density o a key set. If m_c, m_g, v_c and v_g are all small, the keys are scattered throughout the range rather evenly and a distributive method probably will have little advantage over a randomizing method. With a large m_c, a distributive method is expected to do well because the key distribution here resembles the set of keys used to define the characteristics of a transformation. The parameter gap length is particularly important to the distributive methods where a wrong choice of parameters may result in many records being mapped to the same address. This can happen easily in the division method. For example, if v_g is very small, one must select a divisor in the division method not equal to m_g in order to avoid an excessive number of records from going to the same address. The third parameter, density of a key set, has great influence in altering the performance characteristics of a transformation method. For instance, if d is large, the folding method can behave almost like the division method.

The discussion of the categorization of transformation methods and the classification of key set distribution suggests a further experiment to test the various conjectures. In this experiment, the transformation methods should first be characterized with the use of the parameters of order length and collision length. Then we proceed to obtain the parameters m_c, m_g, v_c, v_g and d from the key sets, which ideally should include some files with simple underlying distributions as well as some files with arbitrary underlying distribution. Applying each method to the key sets, we may be able to derive from the results the correlation between the two sets of parameters with respect to performance. If this can be done, then we know how to select a transformation method and associated parameters whenever some simple statistics of a key set are available. In the absence of any statistics, we can always simply use the division method with an arbitrary divisor as suggested earlier.

REFERENCES

(1) Buchholz, W., "File Organization and Addressing," IBM Systems Journal, Vol. 2, 1963.

(2) Lin, A. D., "Key Addressing of Random Access Memories by Radix Transformation," Proceedings, Spring Joint Computer Conference, 1963.

(3) Raver, N. and Schay, G., "A Method for Key-to-Address Transformation," IBM Journal of Research & Development, Vol. 7, April, 1963.

(4) Hanan, M. and Palermo, F. P., "An Application of Coding Theory to a File Addressing Problem," IBM Journal of Research & Development, Vol 7, April, 1963.

(5) Peterson, W. W., "Addressing for Random-Access Storage," IBM Journal of Research & Development, April, 1957.

(6) Schay, G. and Spruth, W. G., "Analysis of a File Addressing Method," Communications of the ACM, August, 1962.

(7) McIlroy, M. D., "A Variant Method of File Searching," Communications of the ACM, March, 1963.

(8) "Introduction to IBM System/360, Direct Access Storage Devices and Organization Methods," IBM Student Text, C20-1649.

(9) Mauer, W. D., "An Improved Hash Code for Scattered Storage," ACM Communications, January, 1968.

(10) Morris, R., "Scatter Storage Techniques," ACM Communications, January, 1968.

(11) Olson, C. A., "Random Access File Organization for Indirectly Addressed Records," Proceedings of ACM National Conference, 1969.

(12) Peterson, W. W., "Error Correcting Codes," M.I.T. Press, Cambridge, Mass., 1961.

(13) Van der Waerden, B. L., "Modern Algebra," Frederick Unger Publishing Co. 1953.

(14) Tainiter, M., "Addressing for Random-Access Storage with Multiple Bucket Capacities," ACM Journal, July, 1963.

(15) Radke, C. E., "The Use of Quadratic Residue Research," ACM Communications, February, 1970.

(16) Bell, J. R., "The Quadratic Quotient Method: A Hash Code Eliminating Secondary Clustering," ACM Communications, February, 1970.

LOAD
FACTOR
0.50

Bucket size / method	1		2		5		10		20		50	
FOLDB	1.39	22.97	1.36	13.34	1.06	1.06	1.01	1.01	1.05	1.01	1.03	1.01
	0.49	-	0.67	-	0.07	0.07	0.01	0.01	0.12	0.03	0.08	0.02
FOLDS	1.33	21.75	1.24	14.09	1.04	1.07	1.01	1.01	1.00	1.00	1.00	1.00
	0.21	-	0.22	-	0.04	0.09	0.02	0.02	0.00	0.00	0.00	0.00
MIDSQ	1.26	1.73	1.14	1.20	1.03	1.03	1.01	1.00	1.00	1.00	1.00	1.00
	0.03	-	0.01	-	0.01	0.01	0.01	0.01	0.00	0.00	0.00	0.00
DA	1.35	4.55	1.24	2.38	1.08	1.08	1.04	1.02	1.05	1.01	1.01	1.00
	0.27	-	0.22	-	0.12	0.14	0.09	0.06	0.05	0.03	0.02	0.00
LIN	1.42	5.15	1.26	3.22	1.06	1.06	1.04	1.02	1.00	1.00	1.00	1.00
	0.48	-	0.32	-	0.09	0.06	0.11	0.04	0.00	0.00	0.00	0.00
DIVISION	1.19	4.52	1.09	1.17	1.02	1.02	1.00	1.00	1.00	1.00	1.00	1.00
	0.09	-	0.06	-	0.02	0.02	0.01	0.01	0.00	0.00	0.00	0.00
GF(16)	1.27	13.61	1.14	3.85	1.03	1.03	1.00	1.00	1.00	1.00	1.00	1.00
	0.08	-	0.04	-	0.01	0.01	0.01	0.00	0.00	0.00	0.00	0.00
GF(2)	1.25	4.00	1.14	1.15	1.03	1.03	1.00	1.00	1.00	1.00	1.00	1.00
	0.06	-	0.12	-	0.01	0.01	0.01	0.00	0.00	0.00	0.00	0.00
Overall	1.31	8.70	1.20	4.49	1.04	1.05	1.01	1.01	1.01	1.00	1.01	1.00
	0.07	-	0.08	-	0.02	0.02	0.02	0.01	0.02	0.01	0.01	0.00
Peterson		1.54		1.24		1.04		1.01		1.00		1.00

*For each method the first row is the average number of accesses per record and the second row the standard error. For each bucket size, the first column is the result of chaining overflow and the second the result of open addressing.

TABLE II

LOAD
FACTOR
.55

Bucket size / method	1		2		5		10		20		50	
FOLDB	1.38	17.18	1.19	2.81	1.04	1.09	1.01	1.01	1.04	0.01	0.07	1.01
	0.34	-	0.18	-	0.04	0.14	0.01	0.00	0.10	0.04	0.19	0.03
FOLDS	1.36	18.32	1.17	4.45	1.04	1.06	1.02	1.02	1.00	1.00	1.00	1.00
	0.17	-	0.11	-	0.04	0.09	0.03	0.04	0.00	0.00	0.00	0.00
MIDSQ	1.30	3.10	1.18	1.45	1.05	1.04	1.01	1.01	1.01	1.00	1.00	1.00
	0.03	-	0.06	-	0.01	0.01	0.01	0.01	0.02	0.01	0.00	0.00
DA	1.39	7.42	1.20	3.02	1.08	1.08	1.04	1.02	1.08	1.02	1.05	1.01
	0.25	-	0.20	-	0.11	0.11	0.08	0.03	0.12	0.04	0.14	0.03
LIN	1.58	6.22	1.45	2.26	1.19	1.17	1.03	1.02	1.02	1.01	1.02	1.00
	0.67	-	0.66	-	0.32	0.26	0.05	0.03	0.06	0.01	0.05	0.01
DIVISION	1.24	5.37	1.12	1.38	1.03	1.03	1.01	1.01	1.05	1.01	1.00	1.00
	0.09	-	0.06	-	0.03	0.02	0.01	0.01	0.34	0.05	0.00	0.00
GF(16)	1.27	13.53	1.15	2.27	1.04	1.04	1.01	1.01	1.00	1.00	1.00	1.00
	0.05	-	0.04	-	0.02	0.02	0.01	0.01	0.00	0.00	0.00	0.00
GF(2)	1.27	6.44	1.15	1.64	1.05	1.06	1.01	1.01	1.00	1.00	1.00	1.00
	0.06	-	0.03	-	0.01	0.05	0.01	0.01	0.00	0.00	0.00	0.00
Overall	1.35	9.70	1.20	2.41	1.07	1.07	1.02	1.01	1.03	1.01	1.02	1.01
	0.10	-	0.10	-	0.05	0.04	0.01	0.01	0.03	0.01	0.03	0.01
Peterson		1.67		1.28		1.05		1.01		1.00		1.00

TABLE III

LOAD
FACTOR
.60

Bucket size / method	1		2		5		10		20		50	
FOLDB	1.44	25.36	1.26	4.56	1.77	18.11	1.40	2.86	1.11	1.03	1.00	1.00
	0.52	-	0.32	-	1.92	45.15	1.02	4.88	0.28	0.07	0.00	0.00
FOLDS	1.52	28.57	1.20	4.64	1.26	16.95	1.05	2.41	1.01	1.00	1.00	1.00
	0.60	-	0.15	-	0.54	41.91	0.07	3.69	0.01	0.01	0.00	0.00
MIDSQ	1.32	3.17	1.19	1.32	1.07	1.07	1.03	1.02	1.01	1.00	1.00	1.00
	0.02	-	0.02	-	0.02	0.01	0.02	0.01	0.01	0.01	0.00	0.00
DA	1.41	8.43	1.29	2.89	1.20	1.17	1.12	1.06	1.18	1.05	1.00	1.00
	0.27	-	0.24	-	0.20	0.18	0.19	0.08	0.33	0.08	0.00	0.00
LIN	1.62	11.50	1.41	3.54	1.21	1.21	1.08	1.04	1.01	1.00	1.00	1.00
	0.65	-	0.48	-	0.22	0.25	0.16	0.09	0.03	0.01	0.00	0.00
DIVISION	1.25	5.04	1.14	2.01	1.05	1.05	1.01	1.01	1.01	1.00	1.00	1.00
	0.10	-	0.06	-	0.03	0.03	0.02	0.01	0.02	0.01	0.00	0.00
GF(16)	1.32	21.36	1.18	3.84	1.06	1.06	1.01	1.01	1.00	1.00	1.00	1.00
	0.07	-	0.05	-	0.05	0.02	0.01	0.01	0.00	0.00	0.00	0.00
GF(2)	1.30	17.97	1.18	2.09	1.06	1.06	1.02	1.01	1.00	1.00	1.00	1.00
	0.07	-	0.05	-	0.02	0.02	0.01	0.01	0.00	0.00	0.00	0.00
Overall	1.40	15.18	1.23	3.09	1.21	5.21	1.09	1.43	1.04	1.01	1.00	1.00
	0.12	-	0.08	-	0.23	7.12	0.12	0.71	0.06	0.02	0.00	0.00
Peter.on		1.82		1.32		1.07		1.02		1.00		1.00

TABLE IV

LOAD FACTOR .65

method \ Bucket size	1		2		5		10		20		50	
FOLDB	1.42	17.76	1.23	6.64	1.29	3.45	1.05	1.03	1.01	1.00	1.00	1.00
	0.34	-	0.53	-	0.49	6.15	0.07	0.03	0.01	0.00	0.00	0.00
FOLDS	1.38	31.55	1.20	4.76	1.11	5.90	1.02	1.02	1.00	1.00	1.00	1.00
	0.18	-	0.13	-	0.09	12.27	0.02	0.02	0.01	0.00	0.00	0.00
MIDSQ	1.34	3.02	1.21	1.41	1.11	1.10	1.04	1.03	1.05	1.03	1.00	1.00
	0.03	-	0.03	-	0.04	0.03	0.22	0.25	0.10	0.07	0.00	0.00
DA	1.42	11.42	1.24	2.91	1.20	1.59	1.06	1.03	1.03	1.01	1.01	1.00
	0.26	-	0.20	-	0.24	0.91	0.09	0.04	0.05	0.01	0.01	0.00
LIN	1.57	7.49	1.48	2.59	1.18	1.24	1.09	1.05	1.12	1.06	1.01	1.00
	0.59	-	0.66	-	0.25	0.22	0.14	0.11	0.38	0.21	0.04	0.01
DIVISION	1.28	8.84	1.16	2.79	1.07	1.10	1.02	1.02	1.00	1.00	1.00	1.00
	0.07	-	0.06	-	0.03	0.09	0.02	0.01	0.01	0.00	0.00	0.00
GF(16)	1.29	21.34	1.20	3.41	1.07	1.43	1.03	1.02	1.01	1.00	1.00	1.00
	0.07	-	0.04	-	0.03	1.36	0.01	0.01	0.01	0.00	0.00	0.00
GF(2)	1.35	25.96	1.21	2.74	1.08	1.10	1.03	1.02	1.01	1.00	1.00	1.00
	0.07	-	0.04	-	0.03	0.07	0.02	0.02	0.01	0.00	0.00	0.00
Overall	1.38	15.92	1.24	3.41	1.14	2.11	1.04	1.03	1.03	1.01	1.00	1.00
	0.09	-	0.09	-	0.07	1.58	0.02	0.01	0.04	0.02	0.01	0.00
Peterson		2.00		1.40		1.10		1.03		1.00		1.00

TABLE V

LOAD FACTOR .70

Bucket size / method	1		2		5		10		20		50	
FOLDB	1.45	23.95	1.30	6.90	1.22	1.93	1.05	1.04	1.17	1.04	1.39	1.04
	0.34	-	0.24	-	0.32	1.92	0.04	0.03	0.45	0.09	0.97	0.12
FOLDS	1.40	39.00	1.29	12.71	1.19	2.14	1.17	1.85	1.08	1.12	1.00	1.00
	0.22	-	0.22	-	0.27	2.51	0.35	2.13	0.16	0.24	0.00	0.00
MIDSQ	1.36	3.85	1.24	1.51	1.12	1.13	1.05	1.04	1.03	1.01	1.01	1.00
	0.03	-	0.04	-	0.04	0.02	0.02	0.02	0.06	0.02	0.02	0.01
DA	1.45	18.87	1.34	5.85	1.26	1.54	1.10	1.06	1.22	1.05	1.39	1.04
	0.29	-	0.24	-	0.24	0.68	0.11	0.06	0.45	0.09	0.99	0.09
LIN	1.63	8.51	1.54	3.74	1.51	1.36	1.19	1.12	1.03	1.01	1.00	1.00
	0.61	-	0.68	-	1.26	0.57	0.38	0.31	0.05	0.02	0.01	0.00
DIVISION	1.28	4.73	1.19	2.71	1.09	1.10	1.03	1.02	1.01	1.01	1.00	1.00
	0.10	-	0.07	-	0.05	0.05	0.03	0.02	0.01	0.01	0.02	0.00
GF(16)	1.33	33.01	1.23	4.35	1.10	1.34	1.04	1.04	1.01	1.01	1.00	1.00
	0.08	-	0.04	-	0.02	0.81	0.02	0.05	0.01	0.01	0.00	0.00
GF(2)	1.36	41.53	1.22	2.46	1.11	1.16	1.05	1.04	1.01	1.01	1.01	1.00
	0.07	-	0.06	-	0.03	0.09	0.02	0.02	0.01	0.01	0.03	0.01
Overall	1.41	21.68	1.29	5.0[illegible]	1.20	1.46	1.09	1.15	1.07	1.03	1.10	1.01
	0.10	-	0.10	-	0.13	0.36	0.06	0.27	0.08	0.04	0.17	0.02
Peterson		2.26		1.52		1.13		1.04		1.01		1.00

TABLE VI

LOAD
FACTOR
.75

Bucket size / method	1		2		5		10		20		50	
FOLDB	1.57	48.70	1.64	29.03	1.22	1.63	1.19	1.08	1.16	1.07	1.89	1.10
	0.64	-	1.11	-	0.29	0.90	0.37	0.10	0.40	0.16	1.56	0.17
FOLDS	1.48	65.10	1.41	29.23	1.15	2.30	1.05	1.05	1.03	1.02	1.02	1.01
	0.24	-	0.34	-	0.09	1.16	0.04	0.03	0.03	0.02	0.05	0.02
MIDSQ	1.40	9.75	1.29	1.80	1.16	1.20	1.09	1.06	1.03	1.02	1.03	1.00
	0.03	-	0.03	-	0.04	0.05	0.06	0.04	0.02	0.01	0.05	0.01
DA	1.49	30.62	1.39	7.41	1.32	3.09	1.26	1.13	1.04	1.02	2.07	1.07
	0.28	-	0.25	-	0.33	1.63	0.37	0.12	0.07	0.03	1.78	0.12
LIN	1.64	10.24	1.60	4.81	1.25	1.67	1.22	1.10	1.04	1.02	1.01	1.00
	0.61	-	0.72	-	0.27	1.63	0.18	0.08	0.08	0.02	0.01	0.01
DIVISION	1.31	7.20	1.19	3.35	1.11	1.29	1.07	1.04	1.03	1.01	1.00	1.00
	0.10	-	0.08	-	0.05	0.27	0.07	0.04	0.03	0.01	0.01	0.00
GF(16)	1.37	45.19	1.25	5.10	1.13	1.81	1.05	1.07	1.02	1.01	1.00	1.00
	0.05	-	0.05	-	0.04	2.09	0.03	0.08	0.01	0.01	0.01	0.00
GF(2)	1.37	56.38	1.24	4.94	1.13	1.30	1.06	1.07	1.02	1.01	1.01	1.01
	0.10	-	0.08	-	0.06	0.29	0.04	0.05	0.02	0.01	0.04	0.01
Overall	1.45	34.15	1.38	10.71	1.18	1.79	1.12	1.08	1.05	1.02	1.25	1.02
	0.10	-	0.16	-	0.07	0.59	0.07	0.03	0.04	0.02	0.31	0.04
Peterson		2.67		1.70		1.20		1.08		1.02		1.00

TABLE VII

LOAD
FACTOR
.80

Bucket size / method	1		2		5		10		20		50	
FOLDB	1.51	41.10	1.45	16.69	1.27	1.51	1.25	1.22	1.34	1.23	1.01	1.00
	0.38	-	0.54	-	0.33	0.66	0.52	0.18	0.83	0.43	0.01	0.01
FOLDS	1.48	66.26	1.37	21.47	1.16	1.70	1.10	1.16	1.04	1.03	1.00	1.00
	0.17	-	0.16	-	0.10	0.71	0.07	0.13	0.03	0.03	0.01	0.00
MIDSQ	1.41	13.25	1.31	2.80	1.19	1.27	1.12	1.10	1.08	1.02	1.01	1.00
	0.03	-	0.04	-	0.05	0.06	0.06	0.04	0.07	0.03	0.01	0.01
DA	1.51	49.72	1.42	8.49	1.34	1.90	1.30	1.26	1.38	1.09	1.05	1.02
	0.26	-	0.27	-	0.35	1.28	0.16	0.34	0.82	0.13	0.07	0.02
LIN	1.81	9.97	1.73	6.39	1.83	1.90	1.35	1.33	1.12	1.05	1.15	1.07
	0.74	-	0.79	-	1.59	1.12	0.50	0.38	0.19	0.06	0.32	0.16
DIVISION	1.34	10.10	1.26	4.84	1.13	1.31	1.08	1.08	1.04	1.03	1.01	1.00
	0.07	-	0.06	-	0.07	0.20	0.05	0.04	0.03	0.02	0.01	0.00
GF(16)	1.41	76.04	1.29	9.04	1.17	2.76	1.09	1.17	1.05	1.03	1.01	1.00
	0.06	-	0.04	-	0.05	18.80	0.04	0.44	0.03	0.02	0.01	0.00
GF(2)	1.40	95.14	1.27	12.07	1.14	1.46	1.09	1.12	1.04	1.04	1.01	1.00
	0.10	-	0.05	-	0.07	0.44	0.66	0.11	0.04	0.07	0.03	0.01
Overall	1.48	45.20	1.39	10.22	1.28	1.73	1.17	1.18	1.14	1.07	1.03	1.01
	0.14	-	0.15	-	0.22	0.45	0.10	0.08	0.13	0.07	0.05	0.02
Peterson		3.24		1.92		1.28		1.11		1.03		1.01

TABLE VIII

LOAD
FACTOR
.85

Bucket size / method	1		2		5		10		20		50	
FOLDB	1.49	54.88	1.34	7.37	1.18	2.12	1.36	1.33	1.04	1.21	1.02	1.01
	0.35	-	0.17	-	0.05	1.31	0.32	0.44	0.05	0.40	0.02	0.01
FOLDS	1.47	67.74	1.34	10.14	1.17	1.65	1.13	1.20	1.06	1.05	1.01	1.01
	0.20	-	0.17	-	0.09	0.39	0.08	0.10	0.05	0.03	0.02	0.01
MIDSQ	1.45	21.82	1.34	4.03	1.22	1.44	1.18	1.18	1.09	1.07	1.04	1.01
	0.03	--	0.04	-	0.02	0.13	0.08	0.08	0.04	0.04	0.01	0.01
DA	1.53	59.95	1.38	8.71	1.24	2.09	1.50	1.41	1.13	1.06	1.05	1.01
	0.28	-	0.22	-	0.18	1.12	0.60	0.45	0.16	0.06	0.16	0.01
LIN	1.81	32.39	1.80	18.10	1.42	1.81	1.24	1.24	1.04	1.04	1.04	1.01
	0.72	-	0.97	-	0.44	0.65	0.22	0.15	0.06	0.03	0.04	0.01
DIVISION	1.41	13.27	1.32	6.55	1.20	2.44	1.25	1.21	1.29	1.08	1.02	1.01
	0.23	-	0.26	-	0.08	2.19	0.60	0.18	1.08	0.18	0.05	0.01
GF(16)	1.41	101.80	1.31	16.56	1.20	2.81	1.10	1.22	1.07	1.06	1.03	1.01
	0.06	-	0.06	-	0.05	4.10	0.06	0.15	0.04	0.03	0.02	0.01
GF(2)	1.43	152.64	1.31	27.81	1.20	1.94	1.12	1.20	1.07	1.07	1.02	1.01
	0.10	-	0.08	-	0.07	1.08	0.06	0.14	0.04	0.04	0.03	0.01
Overall	1.45	63.06	1.39	12.40	1.23	2.04	1.24	1.25	1.10	1.08	1.03	1.01
	0.13	-	0.16	-	0.08	0.32	0.13	0.07	0.08	0.05	0.01	0.00
Peterson		4.15		2.42		1.44		1.17		1.07		1.02

TABLE IX

LOAD
FACTOR
.90

Bucket size / method	1		2		5		10		20		50	
FOLDB	1.55	69.63	1.39	18.83	1.22	4.25	2.98	3.36	1.24	1.27	1.66	1.13
	0.33	-	0.18	-	0.9	5.78	1.65	5.29	0.21	0.38	1.61	0.29
FOLDS	1.40	77.01	1.36	21.01	1.24	4.84	1.42	3.19	1.10	1.17	1.02	1.01
	0.20	-	0.12	-	0.10	2.99	0.58	4.31	0.07	0.15	0.02	0.01
MIDSQ	1.45	27.14	1.37	6.54	1.27	1.81	1.28	1.30	1.15	1.11	1.07	1.03
	0.02	-	0.02	-	0.04	0.26	0.11	0.13	0.03	0.04	0.03	0.02
DA	1.52	89.20	1.41	21.63	1.25	3.86	1.62	1.75	1.17	1.14	1.39	1.11
	0.25	-	0.21	-	0.18	3.63	0.69	0.66	0.15	0.13	0.95	0.25
LIN	1.80	28.69	1.66	15.12	1.44	3.81	1.28	2.89	1.20	1.14	1.08	1.03
	0.66	-	0.82	-	0.42	3.67	0.20	3.48	0.20	0.13	0.08	0.03
DIVISION	1.38	22.42	1.30	4.80	1.24	1.94	1.16	1.32	1.09	1.08	1.03	1.01
	0.10	-	0.10	-	0.09	0.72	0.08	0.20	0.06	0.05	0.03	0.01
GF(16)	1.50	149.87	1.34	27.92	1.24	3.48	1.19	1.52	1.12	1.12	1.04	1.02
	0.07	-	0.05	-	0.06	5.03	0.10	0.93	0.05	0.10	0.03	0.02
GF(2)	1.46	218.47	1.35	49.25	1.23	2.77	1.16	1.36	1.06	1.1[illegible]	1.04	1.02
	0.09	-	0.08	-	0.07	2.06	0.07	0.23	0.07	0.13	0.03	0.03
Overall	1.51	85.30	1.40	20.64	1.27	3.35	1.51	2.09	1.14	1.15	1.17	1.05
	0.12	-	0.10	-	0.07	1.74	0.57	0.84	0.06	0.05	0.22	0.04
Peterson		5.50		3.40		1.76		1.33		1.13		1.03

TABLE X

LOAD
FACTOR
.95

method \ Bucket size	1		2		5		10		20		50	
FOLDB	1.51	97.56	1.38	36.62	1.28	5.59	1.50	1.91	1.68	1.50	1.24	1.12
	0.32	-	0.16	-	0.08	3.50	0.75	0.55	1.46	0.46	0.31	0.12
FOLDS	1.51	118.57	1.40	41.00	1.34	4.81	1.25	3.08	1.14	1.29	1.17	1.12
	0.17	-	0.15	-	0.23	2.47	0.11	2.55	0.09	0.32	0.20	0.15
MIDSQ	1.47	37.53	1.39	10.80	1.32	2.62	1.28	1.67	1.28	1.29	1.15	1.08
	0.04	-	0.04	-	0.02	0.62	0.11	0.28	0.20	0.13	0.07	0.04
DA	1.52	125.59	1.42	38.00	1.32	8.03	1.59	2.74	1.76	1.42	2.17	1.23
	0.26	-	0.21	-	0.19	9.01	0.83	2.06	1.46	0.32	2.52	0.39
LIN	1.44	42.63	1.38	22.25	1.24	3.10	1.20	1.81	1.13	1.20	1.06	1.04
	0.10	-	0.08	-	0.08	1.90	0.06	0.49	0.10	0.16	0.07	0.05
DIVISION	1.41	25.79	1.34	10.80	1.25	4.47	1.20	2.32	1.17	1.25	1.08	1.03
	0.10	-	0.09	-	0.08	4.18	0.08	1.74	0.07	0.17	0.08	0.04
GF(16)	1.48	20.45	1.37	46.46	1.30	7.29	1.22	2.05	1.18	1.35	1.10	1.09
	0.06	-	0.06	-	0.05	8.39	0.06	1.24	0.07	0.34	0.06	0.06
GF(2)	1.48	283.70	1.39	69.42	1.27	7.03	1.21	1.87	1.17	1.35	1.13	1.15
	0.10	-	0.08	-	0.09	7.70	0.08	0.59	0.10	0.21	0.12	0.39
Overall	1.48	93.98	1.38	34.42	1.29	5.37	1.31	2.18	1.31	1.33	1.26	1.11
	0.04	-	0.02	-	0.03	1.85	0.14	0.45	0.24	0.09	0.35	0.06
Peterson		11.0		5.11		2.47		1.76		1.33		1.11

TABLE XI

LOADING
FACTOR
0.50

Bucket size / method	1	2	5	10	20	50
FOLDB	25	15	3	1	1	1
	18	18	3	1	3	1
FOLDS	24	14	3	1	0	0
	12	11	3	1	0	0
MIDSQ	22	11	2	0	0	0
	1	1	1	0	0	0
DA	23	13	3	1	1	0
	12	8	3	2	2	0
LIN	26	13	3	1	0	0
	14	12	4	4	0	0
DIVISION	17	7	2	0	0	0
	7	4	1	1	0	0
GF(16)	23	11	2	0	0	0
	5	3	1	0	0	0
GF(2)	21	10	2	0	0	0
	7	4	1	0	0	0
Overall	23	12	3	1	0	0
	3	2	1	1	1	0
Olson	33	20	7	2	0.2	0

*For each method the first row is the average percentage of overflow records and the second the standard error.

TABLE XII

LOADING
FACTOR
0.55

method \ Bucket size	1	2	5	10	20	50
FOLDB	26	13	3	1	1	1
	14	10	2	1	4	3
FOLDS	27	12	2	1	0	0
	8	5	2	1	0	0
MIDSQ	25	13	3	1	0	0
	2	3	1	1	0	0
DA	27	11	3	1	2	3
	9	7	3	1	3	3
LIN	30	19	7	2	1	0
	19	17	9	2	1	1
DIVISION	21	10	3	1	1	0
	6	4	2	1	4	0
GF(16)	23	11	3	1	0	0
	5	3	1	1	0	0
GF(2)	24	12	3	1	0	0
	7	3	1	1	0	0
Overall	25	13	3	1	1	1
	3	3	1	1	1	1
Olson	36	23	9.5	2.5	0.5	0

TABLE XIII

LOADING
FACTOR
0.60

Bucket size / method	1	2	5	10	20	50
FOLDB	27	15	11	6	3	0
	18	14	20	13	7	0
FOLDS	28	14	9	4	0	0
	12	7	13	6	1	0
MIDSQ	26	14	5	1	0	0
	1	1	1	1	1	0
DA	27	16	8	4	3	0
	11	9	7	6	6	0
LIN	35	20	8	3	0	0
	17	13	8	4	1	0
DIVISION	20	11	3	1	0	0
	6	4	2	1	0	0
GF(16)	26	14	4	1	0	0
	5	3	1	1	0	0
GF(2)	25	13	3	1	0	0
	7	4	2	1	0	0
Overall	27	15	6	3	1	0
	4	2	3	2	1	0
Olson	38	25	12	4	1	0

TABLE XIV

LOADING
FACTOR
0.65

method \ Bucket size		1	2	5	10	20	50
FOLDB		29	16	9	3	0	0
		14	10	13	3	0	0
FOLDS		29	13	6	1	0	0
		9	6	5	1	0	0
MIDSQ		27	15	6	3	1	0
		2	2	1	4	2	0
DA		28	14	8	2	1	0
		9	6	9	2	1	0
LIN		33	21	8	3	2	0
		16	15	7	4	5	1
DIVISION		27	13	5	1	0	0
		5	3	2	1	0	0
GF(16)		26	15	5	2	0	0
		4	3	2	1	0	0
GF(2)		27	15	5	1	0	0
		6	4	2	1	1	0
Overall		28	15	7	2	1	0
		2	2	2	1	1	0
Olson		40	27	13.5	6	2	0

TABLE XV

LOADING
FACTOR
0.70

Bucket size / method	1	2	5	10	20	50
FOLDB	31	19	10	3	4	4
	14	11	9	2	9	9
FOLDS	29	18	8	4	2	0
	9	7	7	6	3	0
MIDSQ	29	17	7	3	1	0
	2	2	1	1	2	0
DA	30	20	11	3	4	4
	10	9	8	3	9	9
LIN	36	24	11	5	1	0
	15	14	14	7	1	0
DIVISION	25	15	6	2	0	0
	7	4	3	1	1	0
GF(16)	28	16	6	2	1	0
	3	2	2	1	1	0
GF(2)	28	16	6	2	1	0
	6	5	2	1	1	0
Overall	30	18	8	3	2	1
	3.	3	2	1	1	1
Olson	42	30	15	8	3	0.2

TABLE XVI

LOADING
FACTOR
0.75

Bucket size / method	1	2	5	10	20	50
FOLDB	33	24	10	6	2	6
	17	19	9	10	5	11
FOLDS	33	23	8	3	1	0
	12	11	3	1	1	1
MIDSQ	31	20	9	4	2	0
	1	1	2	2	1	1
DA	31	21	13	8	1	6
	9	9	10	10	1	11
LIN	36	36	11	5	2	0
	16	20	7	4	2	0
DIVISION	25	16	7	3	1	0
	7	5	2	2	1	0
GF(16)	30	18	8	3	1	0
	3	3	2	1	1	0
GF(2)	29	17	8	3	1	0
	6	5	3	2	1	0
Overall	31	23	9	4	1	2
	3	6	2	2	1	3
Olson	44	32	18	10	4	0.6

TABLE XVII

LOADING
FACTOR
0.80

method \ Bucket size	1	2	5	10	20	50
FOLDB	34	33	12	8	6	0
	14	14	10	11	12	1
FOLDS	35	23	9	5	2	0
	8	7	4	2	1	0
MIDSQ	32	21	10	6	3	0
	1	2	2	2	2	1
DA	34	24	14	9	6	1
	9	9	10	11	12	1
LIN	41	30	17	9	3	2
	17	15	16	9	4	3
DIVISION	28	19	8	4	2	0
	5	3	4	2	1	0
GF(16)	31	20	10	5	2	0
	3	2	2	1	1	0
GF(2)	31	19	9	4	2	0
	6	5	3	2	1	0
Overall	33	22	11	6	3	0
	4	3	3	2	2	1
Olson	45	34	20	12.5	6	1.5

TABLE XVIII

LOADING
FACTOR
0.85

Bucket size / method	1	2	5	10	20	50
FOLDB	34	22	11	11	3	0
	14	8	2	12	1	1
FOLDS	33	22	10	6	3	0
	8	6	3	3	2	1
MIDSQ	34	22	12	7	3	1
	2	2	1	2	1	0
DA	34	22	11	12	4	1
	11	5	4	12	3	1
LIN	42	31	16	8	3	1
	16	16	9	5	1	1
DIVISION	30	21	11	7	4	1
	10	8	3	8	9	1
GF(16)	32	21	11	6	3	1
	3	3	2	2	1	1
GF(2)	32	21	11	6	3	1
	6	4	3	2	1	1
Overall	34	23	12	8	3	1
	3	3	2	2	1	1
Olson	46	36	23	15	8	3

TABLE XIX

LOADING
FACTOR
0.90

method \ Bucket size	1	2	5	10	20	50
FOLDB	37	26	12	15	5	4
	12	8	3	14	4	7
FOLDS	36	26	13	11	4	1
	7	5	3	8	2	1
MIDSQ	34	24	14	9	5	2
	1	1	1	2	1	1
DA	35	23	11	15	4	3
	7	6	4	13	3	6
LIN	45	33	20	11	6	2
	17	15	13	6	4	1
DIVISION	29	21	11	7	3	1
	6	5	3	2	2	1
GF(16)	34	23	13	8	4	1
	2	3	2	2	1	1
GF(2)	33	23	12	7	4	1
	5	4	2	2	2	1
Overall	37	24	13	10	4	2
	5.	4	3	3	1	1
Olson	48	38	25	17.5	11	5

TABLE XX

LOAD
FACTOR
0.95

Bucket size / method	1	2	5	10	20	50
FOLDB	36	25	15	14	8	4
	12	7	3	13	4	3
FOLDS	36	24	15	9	5	3
	6	5	5	3	2	3
MIDSQ	35	25	16	11	7	3
	2	2	1	2	3	1
DA	35	25	13	15	11	5
	7	6	4	13	14	6
LIN	41	32	19	10	8	3
	16	17	13	5	9	3
DIVISION	33	23	14	8	6	2
	6	4	3	2	2	2
GF(16)	36	24	14	9	6	2
	.	3	2	2	2	1
GF(2)	35	24	14	9	6	3
	4	4	3	2	2	2
Overall	36	25	15	11	6	3
	2.	3	2	2	2	1
Olson	49	39	26	20	14	7

TABLE XXI

SECTION II

A QUANTITATIVE APPROACH TO THE SELECTION OF SECONDARY INDEXES

F. P. Palermo

A QUANTITATIVE APPROACH TO THE SELECTION OF SECONDARY INDEXES *

by

F. P. Palermo

Information Sciences Department
IBM Research Laboratory
San Jose, California

ABSTRACT: In this report, formal definitions of the concepts of secondary index, key index, query and query load are given. This is done for the case of a single relation (a subset of the cartesian product of a number of domains). The definitions are used to formulate a problem in secondary indexes and show how the concept of query load is related to the concept of secondary indexes. An evaluation criterion is formulated which pinpoints the kind of input data that is needed to evaluate various selections of secondary indexes to match the query load.

INTRODUCTION

In information retrieval systems, data is stored on the peripheral storage devices in several possible ways. One method is to divide the data into records, each of which has a unique identifier called the primary key and to physically store these records so that a record can be easily retrieved if the key for that record is given. However, if a request is made to retrieve a record by giving the value of a particular attribute other than the key attribute, all of the data has to be retrieved and examined in order to respond to the request. In order to make this kind of retrieval more effective, an auxiliary table may be created which either directly gives the addresses of those records that have specified values for the given attribute, or indirectly to give the list of keys for those records having the given attribute value. When such a table is created, we say that a secondary index has been created for the data.

Clearly, this added retrieval capability becomes more desirable as the number of requests for records using this attribute increases. However, the price to be paid for this capability is the added amount of space required for the storage of the table. Requests for retrieval of records are called queries. If a number of different attributes arise in the queries which refer to this data, the cost of storing the additional secondary indexes to accommodate the attributes increases. The problem of secondary indexes can then be phrased as follows: In view of all the queries on the data, what set of secondary indexes should be selected to facilitate the retrieval and keep the storage costs down.

In the following sections, formal definitions are given for the concepts of key index, secondary index, query and query load in a set-theoretical framework. We restrict our considerations to a single relational set R which is defined as a subset of the cartesian product of a number of domains. This formalism enables us to define the concepts of key index and secondary index in terms of partitions of R into subsets. It also enables us to give unambiguously the essential features of a query which include the subset to be retrieved.

Once these notions are clearly specified, a quantitative measure of compatibility between index induced partitions of R and the pertinent subset of data specified by a query is introduced. An evaluation criterion, in terms of these compatibility measures and the parameters extracted from the query load, can then be established to aid in the selection of secondary indexes.

1. SECONDARY INDEXES AND PARTITIONS

In this section, we consider a relation R as a subset of the cartesian product of elementary domains A_i, $i = 0, 1, \ldots, k$.

Thus $$R \subseteq A = A_0 \times A_1 \times \ldots \times A_k .$$

For each i, we consider the projection of the cartesian product A onto the factor A_i. This projection defines a function $\Pi_i : R \to A_i$ which we call the projection of R into A_i.

In the case where $\Pi_i : R \to A_i$ is $1 : 1$, we say that the domain A_i is a key domain. Thus a domain is a key domain for the relation R if the projection satisfies the condition: If $\Pi_i(r_1) = \Pi_i(r_2)$, then $r_1 = r_2$.

We shall assume that A_0 is a key domain for the relation R. We let $R_0 \subset A_0$ be the image of R under the projection Π_0. Thus $\Pi_0 : R \to R_0$ establishes a one-to-one correspondence between the elements of R and the elements of R_0 which are called the keys for R.

Since Π_0 is an isomorphism between R and R_0, we shall, in the following discussion, deal with R. However, the entire development can be done in terms of R_0.

We introduce the concept of a partition of R. We say that a collection $\mathscr{P}(R)$ of subsets of R forms a partition of R if:

(1) For B_1 and B_2 in $\mathscr{P}(R)$ $B_1 \cap B_2 = \emptyset$ whenever $B_1 \neq B_2$. i.e., the elements of $\mathscr{P}(R)$ are pairwise disjoint.

(2) R is the union of the elements of $\mathcal{P}(R)$. i.e., $R = \bigcap_{B \in \mathcal{P}(R)} B$.

We now show that partitions of R arise in very natural ways. Let $L = \{0,1,\ldots,k\}$ and let $A = A_0 \times A_1 \times \ldots \times A_k$ be the domain of R, i.e., $R \subseteq A$. For each subset $K \subset L$, we form the cartesian product $A_K = \prod_{i \in K} A_i$ and let $\Pi_K : R \to A_K$ be the projection function from R to A_K.

For each a in A_K, we let $R(a, K) = \{r \in R | \Pi_K(r) = a\}$ and let $\mathcal{P}_K(R) = \{R(a, K) | a \in A_K\}$. It is easy to see that $\mathcal{P}_K(R)$ is a partition of R. We call this the partition of R induced by the projection Π_K.

If $K = \{k\}$, we write $\mathcal{P}_k(R)$ instead of $\mathcal{P}_K(R)$.

If $\mathcal{P}_{K_1}$ and $\mathcal{P}_{K_2}$ are two partitions of R, we define the intersection of the partitions denoted $\mathcal{P}_{K_1} \cap \mathcal{P}_{K_2}$ to be that partition of R consisting of all sets of the form $A \cap B$ where $A \in \mathcal{P}_{K_1}$ and $B \in \mathcal{P}_{K_2}$.

It is easy to extend this concept of intersection of partitions to a family of partitions. We can show that if $K \subset L$, then $\mathcal{P}_K$, the partition induced on R by Π_K is the intersection of the partition $\mathcal{P}_i$ for $i \in K$.

Thus $\mathcal{P}_K = \bigcap_{i \in K} \mathcal{P}_i$.

We are now in a position to define the concept of secondary index. Let $\mathcal{P}_K$ be the partition of R induced by the projection $\Pi_K : R \to A_K$. The function $\phi_K : A_K \to \mathcal{P}_K$ defined by $\phi_K(a) = R(a, K)$ is called an index of R with respect to A_K.

Thus, we see that an index is a function between the domain A_K and the partition induced by the projection Π_K.

In the case where Π_K is 1 : 1, the partition $\mathcal{P}_K(R)$ is in 1 : 1 correspondence with the elements of R and the corresponding index ϕ_K may serve as a key index.

2. QUERIES AND THEIR FORMAL DEFINITIONS

In this section, we shall formulate definitions for a query and show how they relate to the concepts of index and partition defined in section 1.

First, we observe that given a relation R, a query is a request for the values in a selected set of domains for a subset of R. The subset of R is specified by giving a qualifying expression.

Thus, for example, a query can take the form: What is the value in domain A_3 for all elements of R which have a value b in domain A_1? Thus, a query Q can be thought of as having two parts, a specification part and an output part. Formally, we write $Q = (Q_S, Q_0)$. Here Q_S is the specification part of the query and defines a subset of R. Q_0 is the output part of the query and specifies which domains are required to be displayed for the subset of R specified by Q_S. Thus, Q_0 must specify which domains are to be displayed. This can be effectively achieved in the case where the relation R is a subset of A, the cartesian product of the set of domains $\{A_i | i = 0,1,\ldots,k\}$. If we let $L = \{0,1,\ldots,k\}$, then Q_0 can be specified as a subset of L. i.e., $Q_0 \subset L$. For the remainder of this paper, we shall be concerned with the specification part of the query.

The function of the specification Q_S of the query Q is to determine a particular subset of R. In order to do this, observe that the only means we have to use in the specification of a subset of R are through the use of the projections of R to the elementary domains. Thus, we let $p(a,i)$ stand for the expression $\Pi_i(r) = a$. The subset of R defined by $p(a,i)$ will be denoted by $R(a,i)$. Thus, we have that $R(a,i) = \{r | \Pi_i(r) = a\}$. We shall call an expression of the form $p(a,i)$ a primitive expression and the corresponding set $R(a,i)$ a basic subset of R. If we compare the definition of $R(a,i)$ with the concept of partition $\mathscr{P}_i(R)$, we see that $R(a,i)$ is an element of the partition $\mathscr{P}_i(R)$. In fact, it is the image of the index $\phi_i : A_i \to \mathscr{P}_i(R)$ evaluated at a, i.e., $R(a,i) = \phi_i(a)$. In this case, we see that the query specification defines a particular element of a partition. We are now in a position to define the scope of Q_S, the specification part of the query. We take Q_S to be a subset of R defined by $Q_S = \{r | P(r)\}$ where $P(r)$ is a well-formed formula of the first order predicate calculus which has no quantifiers and is obtained by using the primitive expressions defined above. We restrict ourselves to this type of query specification in order to avoid the myriad complications which arise by allowing quantifiers.

Now it is well known that every formula $P(r)$ can be transformed into an equivalent formula which is in what is called the disjunctive normal form. We say that a formula is in the disjunctive normal form if it is of the form $P_1 \vee P_2 \vee \ldots \vee P_n$, where each P_i is the conjunction formed from the elementary expressions and their negations.

Thus, $P(r)$, the qualification statement has the form $P_1(r) \vee P_2(r) \vee \ldots \vee P_n(r)$ where each $P_i(r)$ is the conjunction of terms as above. Since each $P_i(r)$ defines a subset of R by the formula $B_i = \{r | P_i(r)\}$, we find that the qualified subset Q_S of the query can be written as the union of the sets B_i. We shall look a little closer at the expression $P_i(r)$.

Let $P_i(r)$ be the conjunction of terms, each of which is either an elementary expression or the negation of an elementary expression. We observe that in this conjunction at most one elementary expression for each domain can occur. Thus, we observe that the elementary domains represented in the formula for $P_i(r)$ form a subset of all the elementary domains and can be characterized by a subset K of $L = \{0,1,\ldots,k\}$.

Thus, $P_i(r) = \bigwedge_{j \varepsilon K} q_j$

where q_j is an elementary expression of one of the forms:

(1) $\Pi_j(r) = a_j$ for some $a_j \varepsilon A_j$

(2) $\Pi_j(r) \neq a_j$ for some $a_j \varepsilon A_j$.

The set K can be divided into two subsets K_+ and K_- as follows:

$$j \varepsilon K_+ \text{ if } \Pi_j(r) = a_j \text{ for } a_j \varepsilon A_j$$

$$j \varepsilon K_- \text{ if } \Pi_j(r) \neq a_j \text{ for } a_j \varepsilon A_j.$$

We then define for each $P_i(r)$, the corresponding conjunction $P_i'(r)$ defined by

$$P_i'(r) = \bigwedge_{j\varepsilon K_+} q_j$$

Clearly, $P_i(r)$ implies $P_i'(r)$ because of the tautology $p \wedge q \rightarrow p$.

In this case, take $p = \bigwedge_{j\varepsilon K_+} q_j$,

$$q = \bigwedge_{j\varepsilon K_-} q_j$$

Then $p \wedge q = \bigwedge_{j\varepsilon K} q_j = P_i(r)$

and the result follows.

Now for each $P_i(r)$, let $P_i'(r)$ be conjunction obtained as above and define $B_i = \{r|P_i(r)\}$ and

$$B_i' = \{r|P_i'(r)\}.$$

Some remarks are in order relative to the set B_i'.

If $K_+ = \emptyset$, i.e., $K_- = K$, then all the conjuncts are negations of elementary expressions. In this case, we take $B_i = R$.

Thus, for each query $Q = (Q_S, Q_0)$ we can assign two families of sets, namely,

$$B_i \quad i = 1,\ldots,n \text{ and } B_i' \quad i = 1,\ldots,n$$

corresponding to the specification expression's conjunctive normal form $P_1(r) \vee P_2(r) \vee \ldots \vee P_n(r)$ where each $P_i(r)$ is the conjunction of elementary terms. Observe that $B_i \subset B_i'$ because $P_i(r)$ implies $P_i'(r)$.

3. QUERY-PARTITION COMPATIBILITY

In this section, we shall introduce a qualitative method for comparing a query with a partition. First we define the notion of a set B being compatible with a partition. Then we extend this concept to a query and finally, to a query load.

Let B be a subset of R and $\mathcal{P}$ a partition of R.

Thus, $\mathcal{P} = \{R_a | a \in \alpha\}$ where the R_a are pairwise disjoint and their union is R.

We say that B is <u>equi-compatible</u> with $\mathcal{P}$ if

$$B = R_a \text{ for some } a \in \alpha ,$$

B is <u>under-compatible</u> with $\mathcal{P}$ if

$$B \subset R_a \text{ for some } a \in \alpha ,$$

and B is <u>over-compatible</u> with $\mathcal{P}$ if

$$B = \bigcup_{a \in \beta} R_a \text{ for some } \beta \subset \alpha .$$

B is said to be <u>in-compatible</u> with $\mathcal{P}$ if none of the above three conditions hold. If B is not incompatible with $\mathcal{P}$, then B is said to be <u>compatible</u> with $\mathcal{P}$.

It is always possible to find a subset γ of α such that $B \subset \bigcup_{a \in \gamma} R_a$. The above definitions of compatibility are used to distinguish the special cases where the subset γ consist of a single element of α or where the inclusion is actually an equality.

Let $\{B_i | i = 1,\ldots,n\}$ be the family of sets corresponding to the query specifications Q_S as given in section 2.

We say that the query $Q = (Q_S, Q_0)$ is compatible with the partition $\mathscr{P}$ if each B_i is compatible with $\mathscr{P}$.

Now that we have defined the compatibility of a query with a partition, we introduce the notion of a query load and extend the concept of compatibility to that of the query load being compatible with the given partition.

The concept of query load is obtained from the intuitive notion of all the queries which are formulated for a period of time for the relation R. Thus, suppose that over a period of time, the set of queries $\{Q_i | i = 1,\ldots,N\}$ are observed, each occurring with a frequency h_i, $i = 1,\ldots,N$. Then this gives some indication of the requirements of the system with respect to questions asked of it for the relation R.

Thus, we shall define a query load $\mathscr{L}$ to be the set of ordered pairs

$\{(Q_i, h_i) | i = 1,\ldots,N\}$, where Q_i is a query and h_i is an integer representing the frequency of occurrence of Q_i.

We could extend the notion of the query load $\mathscr{L}$ to be compatible with the partition $\mathscr{P}$ in the obvious way, i.e., by requiring that each Q_i be compatible with $\mathscr{P}$.

However, this approach leads to a classification of a query load and a partition being compatible which ignores the frequency of a query. Thus, for example, a partition $\mathscr{P}$ may be compatible with all but a single query in the query load and be classified as being incompatible. Thus, we seek to introduce a concept of degree of compatibility.

As a first approximation, we assign some figures to measure the degree of compatibility between a set B and a partition $\mathscr{P}$.

Thus, we assign values c_1, c_2, c_3 and c_4 to the set B depending on its compatibility with the partition $\mathscr{P}$. The assignments made to B are given in Table 1.

Type of Compatibility	Value
Equi-compatible	c_1
Over-compatible	c_2
Under-compatible	c_3
In-compatible	c_4

TABLE I

The values of the c_i's have the further constraint of being ordered. Thus, we postulate that they satisfy the inequalities:

$$c_1 \le c_2 \le c_3 \le c_4 .$$

Intuitively, these inequalities reflect the notion that, for example, it is easier to retrieve B if it is equi-compatible with $\mathscr{P}$, than if it is over-compatible with $\mathscr{P}$.

Thus, to each query we assign a value which is obtained as the sum of the values for the individual sets B_i associated with the query. Some care should be exercised here in the selection of the c_i, but we are interested in a first approximation for the measure of compatibility between a query Q and a partition $\mathscr{P}$. Thus, we can attach a measure of compatibility between a query Q and a partition $\mathscr{P}$ of R which we denote by $c(Q, \mathscr{P})$.

If $\mathscr{L} = \{(Q_i, h_i) | i = 1, \ldots, N\}$ is a query load, then we can define a measure of compabibility between $\mathscr{L}$ and $\mathscr{P}$ denoted by $c(\mathscr{L}, \mathscr{P})$ by the formula

$$c(\mathscr{L}, \mathscr{P}) = \sum_{i=1}^{N} c(Q_i, \mathscr{P}) \cdot h_i \quad .$$

By this procedure, we can compare the various partitions $\mathcal{P}$ with respect to the query load $\mathcal{S}$ and thus, obtain a first order qualitative evaluation of the partitions of R with respect to a given query load.

This measure of compatibility between the query load $\mathcal{L}$ and the partition $\mathcal{P}$ can then be used to evaluate the effectiveness of choosing an index. This follows easily from the definition of index given in section 2. Thus, if L is the set of domains which are to be used as secondary indexes, we consider the partition $\mathcal{P}_L$ corresponding to this set L and evaluate the function $c(\mathcal{P}_L, \mathcal{L})$ to give a figure of merit for the secondary indexes L as related to the query load $\mathcal{L}$.

This framework can be extended in several directions and the above discussion only serves as a first approximation to the selection of a set of secondary indexes which are responsive to a query load. The principal problem which the designer now must consider is the appropriate selection of the values c_1, c_2, c_3 and c_4. These values should reflect the actual physical size of the data stored, and the access times for the retrieval of the atomic particles of the partition being considered.

SECTION III

SOME RESULTS ON STORAGE SPACE REQUIREMENT AND RETRIEVAL TIME IN FORMATTED FILE ORGANIZATION

S. P. Ghosh

SOME RESULTS ON STORAGE SPACE REQUIREMENT AND RETRIEVAL TIME IN FORMATTED FILE ORGANIZATION *

by

S. P. Ghosh

Information Sciences Department
IBM Research Laboratory
San Jose, California

ABSTRACT: Some basic mathematical concepts underlying the interaction between queries and records of a file have been discussed. Formulas, for the storage-space need in file organization when chaining techniques are used, have been derived for Simple formatted files and Hierarchical files. Some simplifications of formulas in special cases have also been discussed. The results of the chaining technique have been compared with that of the Inverted File. The average retrieval time need to retrieve records when chaining techniques are used have been calculated and compared with that of Inverted File organization.

1. INTRODUCTION

In electronic data processing, information is recorded as sequences of binary bits. A collection of infoımation describing an event, a physical object, or any other type of entity is usually called a record. In general, a record is a collection of small information units which represent values for the attribute or properties of the entity. This record may be represented as $(v_1, v_2, \ldots, v_m)$ where the v_i's are information units. Normally each entity is uniquely identified by the value of some attribute or combination of attributes to allow it to be discussed and maintained unambiguously. Sometimes an identification information unit is added to each record or one or more information units in the record may be used as an identifier. The identifier is usually called the key of the record. Thus, if there are N records, they may be represented as:

$$R_i = (k_i, v_{i1}, v_{i2}, \ldots, v_{im_i})$$

where k_i is the key and the v_{ij}'s are the m_i different information units contained in the ith record.

A collection of records is called a file. If the information have some format structure imposed on them then the records are called formatted records; and the file is called a formatted file. In most data processing problems, the files are formatted. In this paper, we will consider only two types of structured files; namely, Simple Formatted Files and Hierarchical Files.

With regard to the contents of formatted records, the value or information unit relating to the same attribute is stored in a fixed position with respect to other information units in the record. This relatively fixed position of a record, and hence of the file, is often referred to as a field. Thus all the v_{ij}'s for a fixed j and i = 1, 2, ..., N may be referred to as values of the jth field or jth attribute. When every field represents a physically distinct attribute and every record contains one value of each attribute, then the file is referred to as a Simple Formatted file. Thus in Simple Formatted file, all the m_i's are equal.

In a large file, the values of an attribute will not always be distinct. There are some attributes for which every record will have a distinct value; but in most of the practical situations, this will not be true. Thus the frequency distribution of values of one or more attributes will be a useful statistic in many problems relating to information storage and retrieval.

The ability to retrieve segments of information, when desired, is one of the most important aspects of storing information in a computer system. The process specifying the subset of the file to be retrieved is called querying. A query essentially specifies a subset of the file by a series of conditional statements, and the retrieval process consists of retrieving all records which satisfy those statements. The information specified by a query may relate to the key field or to values of other field, e.g., Retrieve all records whose keys lie between K' and K" or retrieve all records in which the the j^{th} attribute has the values v_1, or v_2 or v_3 etc.

The query structures imposes a frequency distribution on a file which may be explained as follows. Consider a simple formatted file with attributes $A_1, A_2, \ldots, A_m$. The attribute A_j can take n_j values, $j=1, 2, \ldots, m$. Thus the total number of different types of records possible is $\prod_{j=1}^{m} n_j = n$. Though n types of records are possible, the file may contain fewer records. If the file contains more than n records, then there are some records which are identical with respect to the m attributes $A_1, A_2, \ldots, A_m$; but there are other information units in the record which makes them distinct. Let

$$f(A_1 = v_1, A_2 = v_2, \ldots, A_m = v_m) \qquad (1.1)$$

denote the frequency of the number of records for which the attribute A_1 takes the value v_1, A_2 takes the value $v_2, \ldots,$ A_m takes the value v_m. The frequency functions (1.1) characterize the frequency distribution of the records in the file. If the queries specify values of attributes, then the frequency distribution of queries can be derived from (1.1), e.g., suppose a query specifies

$A_1=v_1$, $A_2=v_2, \ldots A_i=v_i$, then the frequency of this query is given by

$$\sum_{S_i} f(A_1=v_1,\ A_2=v_2, \ldots, A_m=v_m)$$

where S_i is the sum over all values of the attributes A_{i+1}, $A_{i+2}, \ldots, A_m$.

One of the main purposes of organizing records in a file is to reduce the time needed to retrieve the records pertinent to queries. The problem becomes more complex because of some uncontrollable factors like addition of new records, addition of new queries, etc., which have to be taken into account. In a dynamic environment as the size of the file grows, the cost of reorganizing the records becomes large. In most cases, the same record is pertinent to more than one query, hence the problem of additional storage space becomes another restriction on organization. The two important factors, storage-space and retrieval time, act in opposite directions in file organization. Thus, trying to reduce one of these factors leads to increase in the other. The two extreme situations are reflected in the Query Inverted File Organization and Natural Storage Organization.

An appropriate meaning of the term, Query Inverted File, is "to reorganize the records in such a manner that certain types of information units can be regarded as identification units of the records." Thus, a simple formatted file can be inverted with respect to the values of one attribute or combination of values of one or more attributes or with respect to a set of queries. Suppose there are k queries represented by q_1, $q_2, \ldots, q_k$. Then an inverted File with respect to these queries is constructed in the following manner. All records which satisfy the query $q_i (i=1, 2, \ldots, k)$ are stored in adjacent locations, with q_i as the identification label for all these records. It is obvious that if a record qualifies for more than one query, then it will have to be stored more than once. Hence redundant storage-space is needed. As records pertaining to any one query are adjacently stored, retrieval time is minimum. Thus in the Query Inverted File storage-space is sacrificed for retrieval time.

In the Natural Storage Organization, the records are stored in the same order as they are added to the file, i.e., there is no special type of organization. Thus the storage-space needed is minimum. When records pertaining to a query are to be retrieved, the query is matched with every record in the file to determine the pertinent records. Thus the retrieval time is maximum.

Other file organization methods try to balance between storage-space and retrieval time by using other techniques. One of these methods, usually referred to as the chaining technique, will be discussed in details in this paper.

The main concept underlying the chaining technique is to link records pertaining to a query by link fields. The link field contains the location of the next record pertaining to that query. Thus the necessity of scanning all records to find pertinent records can be eliminated. When a record qualifies for more than one query, redundant storage of records can also be avoided. Storage-space for the chaining technique is larger than Natural Storage Organization because of the link fields; and the retrieval time is more than Inverted File because the records are not stored in spacial proximity. In some situations the chaining technique can be very complex because it depends not only on the pertinent records but also on the physical locations of the records in the storage system. There are many methods for reducing complexity of the chaining technique, but this can be achieved only by increasing the retrieval time or storage-space or a combination of both. One such method is grouping records into buckets and then confining chaining to within buckets. Details of such methods will be discussed in the latter part of the paper.

The concepts discussed in this section are fundamental concepts of file organizations. These have been introduced in the field by many researchers, and the original inventors cannot be identified. Hence no attempt has been made to associate literatures in the preceding discussions. However, some references in which these concepts may be traced are: Gray et al (1961), Buchholz (1963), Baker (1963), Davis and Lin (1965), IBM Report (1967), Abraham, Ghosh, and Ray-Chaudhuri (1968), etc.

2. RELEVANT MEASURE-SPACE FOR STORAGE

In this section we shall introduce the structure of the space of events and the measure functions that have to be defined for calculating storage space when the chaining technique of organization is used. Let the set of queries be denoted by $q_1, q_2, \ldots, q_k$. Each record can be classified into two classes with respect to a query; namely, whether the record satisfies the query or not. Without any loss of generality, the symbol q_i can be used to denote the event that a record satisfies the query q_i and $\bar{q}_i$ to denote the event that a record does not satisfy the query q_i. Thus the binary event space of the query q_i can be denoted by $Q_i = \{q_i, \bar{q}_i\}$. Consider the k-dimensional product space denoted by

$$S = Q_1 \times Q_2 \times \ldots \times Q_k.$$

An element of this product space is denoted by $\theta = (\theta_1, \theta_2, \ldots, \theta_k)$ where θ_i can take two values q_i and $\bar{q}_i$. The element θ represents the classification of a record with respect to the set of k queries, eg., if $\theta = (q_1, \bar{q}_2, \bar{q}_3 \ldots, q_k)$, it means that the particular record corresponding to θ satisfies the query q_1 but does not satisfy the queries q_2 and q_3 but satisfies q_k. In order to define any measure over the product space S, sigma-fields (σ-fields) have to be introduced. As each Q_i is a binary field, the σ-field over Q_i contains $\sigma_i = \{q_i, \bar{q}_i, \phi_i, \Omega_i\}$, where ϕ_i is the event that a record has no classification with respect to the query q_i and Ω_i is the event that the record may or may not satisfy the query q_i. Thus the σ-field over S ($\sigma\{S\}$) is the product space of the σ-fields over Q_i's, i.e., $\sigma\{S\} = (\sigma_1, \sigma_2, \ldots, \sigma_k)$. For every $\sigma \varepsilon \sigma\{S\}$, $f(\sigma)$ will denote the number of records in the file for which the compound event σ is satisfied.

<u>Example 2.1</u>

Suppose there are 4 queries denoted by q_1, q_2, q_3, and q_4. Then S contains 2^4 points of the form $\theta = (\theta_1, \theta_2, \theta_3, \theta_4)$ where $\theta_i = q_i$ or $\bar{q}_i$, i=1, 2, 3, 4. $\sigma\{S\}$ contains 4^4 points of the following type:

$$\sigma = (\sigma_1, \sigma_2, \sigma_3, \sigma_4)$$

where σ_i can take any of the four values q_i, $\bar{q}_i$, ϕ_i, and Ω_i for i=1, 2, 3, 4.

Thus an event of the type $(q_1, \bar{q}_2, \phi_3, \Omega_4)$ represents a record which satisfies the query q_1, does not satisfy the query q_2, has no classification with respect to the query q_3, and may or may not satisfy the query q_4. $f(\sigma)$ indicates the number of records in the file which have the above specification.

Simple Formatted Files

In the simple formatted file, any record can be classified into three states with respect to any query; namely, q_i, $\bar{q}_i$, and Ω_i. Thus for a simple formatted file, ϕ_i does not appear in any co-ordinate position of $\sigma\{S\}$, hence, will not be discussed in the present context.

In this representation, the frequency of records pertinent to a query q_i is denoted as

$$f(\Omega_1, \Omega_2, \ldots, \Omega_{i-1}, q_i, \Omega_{i+1}, \ldots, \Omega_k). \tag{2.1}$$

Thus the frequency of the records which are pertinent to the queries $q_{i_1}, q_{i_2}, \ldots, q_{i_i}$ is denoted by

$$f(\Omega_1, \ldots, q_{i_1}, \ldots q_{i_2} \ldots, q_{i_i}, \ldots \Omega_k). \tag{2.2}$$

For calculating the storage-space needed for chaining technique, it will be assumed that the records are of equal length. The length of a record will be denoted by r. In chaining technique, duplicate storage of records is avoided by using link fields. Hence, if the query set is given by $q_1, q_2, \ldots, q_k$, then the frequency of records in the file organization is given by

$$f(B) = f[\bigcup_{i=1}^{k} (\Omega_1, \Omega_2, \ldots, \Omega_{i-1}, q_i, \Omega_{i+1}, \ldots, \Omega_k)]. \tag{2.3}$$

The symbol $f(B)$ is used to represent (2.3) because in many file organizations the chaining is done only within a bucket, hence if $q_1, q_2, \ldots, q_k$ are the queries pertinent to a bucket, then (2.3) gives the frequency of a bucket. (2.3) can be expressed in terms of (2.1) and (2.2) in the following manner:

If exclusive union is denoted by + and difference by −, then

$$(q_1, \Omega_2,\ldots,\Omega_k) \cup (\Omega_1, q_2, \Omega_3,\ldots,\Omega_k)$$

$$= (q_1, \Omega_2,\ldots,\Omega_k) + [(\Omega_1, q_2, \Omega_3,\ldots,\Omega_k)$$

$$- (q_1, \Omega_2,\ldots,\Omega_k) \cap (\Omega_1, q_2, \Omega_3,\ldots,\Omega_k)]$$

$$= (q_1, \Omega_2,\ldots,\Omega_k) + (\bar{q}_1, \Omega_2,\ldots,\Omega_k) \cap (\Omega_1, q_2, \Omega_3,\ldots,\Omega_k).$$

Thus, $$f[(q_1, \Omega_2,\ldots,\Omega_k) \cup (\Omega_1, q_2, \Omega_3,\ldots,\Omega_k)]$$

$$= f(q_1, \Omega_2,\ldots,\Omega_k) + f[(\bar{q}_1, \Omega_2,\ldots,\Omega_k) \cap (\Omega_1, q_2, \Omega_3,\ldots,\Omega_k)]$$

$$= f(q_1, \Omega_2,\ldots,\Omega_k) + f(\Omega_1, q_2, \Omega_3,\ldots,\Omega_k)$$

$$- f(q_1, q_2, \Omega_3,\ldots,\Omega_k) \tag{2.4}$$

Similarly, $$(q_1, \Omega_2,\ldots,\Omega_k) \cup (\Omega_1, q_2, \Omega_3,\ldots,\Omega_k) \cup (\Omega_1, \Omega_2, q_3, \Omega_4,\ldots,\Omega_k)$$

$$= (q_1, \Omega_2,\ldots,\Omega_k) + (\bar{q}_1, \Omega_2,\ldots,\Omega_k) \cap (\Omega_1, q_2, \Omega_3,\ldots,\Omega_k)$$

$$+ (\bar{q}_1, \Omega_2,\ldots,\Omega_k) \cap (\Omega_1, \bar{q}_2, \Omega_3,\ldots,\Omega_k) \cap (\Omega_1, \Omega_2, q_3, \Omega_4,\ldots,\Omega_k).$$

Thus, $$f[q_1, \Omega_2,\ldots,\Omega_k) \cup (\Omega_1, q_2, \Omega_3,\ldots\Omega_k) \cup (\Omega_1, \Omega_2, q_3, \Omega_4,\ldots,\Omega_k)]$$

$$= f(q_1, \Omega_2,\ldots,\Omega_k) + f(\Omega_1, q_2, \Omega_3,\ldots,\Omega_k) + f(\Omega_1, \Omega_2, q_3, \Omega_4,\ldots,\Omega_k)$$

$$- f(q_1, q_2, \Omega_3,\ldots,\Omega_k) - f(q_1, \Omega_2, q_3, \Omega_4,\ldots,\Omega_k)$$

$$- f(\Omega_1, q_2, q_3, \Omega_4,\ldots,\Omega_k) + f(q_1, q_2, q_3, \Omega_4,\ldots,\Omega_k) \tag{2.5}$$

Similarly by induction it can be shown:

$$f(B) = f[\bigcup_{i=1}^{k}(\Omega_1\ldots,\Omega_{i-1}, q_i, \Omega_{i+1},\ldots,\Omega_k)]$$

$$= \sum_{i=1}^{k} f(\Omega_1,\ldots, \Omega_{i-1}, q_i, \Omega_{i+1},\ldots,\Omega_k)$$

$$- \sum_{i_1}^{k} \sum_{\neq i_2}^{k} f(\Omega_1\ldots, \Omega_{i_1-1}, q_{i_1},\ldots q_{i_2},\ldots, \Omega_k)$$

$$+ \sum_{i_1}^{k} \sum_{\neq i_2}^{k} \sum_{\neq i_3}^{k} f(\Omega_1\ldots q_{i_1},\ldots q_{i_2},\ldots q_{i_3},\ldots, \Omega_k)$$

$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$

$$+ (-1)^{k+1} f(q_1, q_2,\ldots, q_k) \tag{2.6}$$

If (2.6) is multiplied by the length of a record, i.e., r, then the space occupied by the records excluding the space occupied by link fields can be obtained. If the length of the link field is assumed to be constant and equal to ℓ, then the space occupied by the link fields is given by

$$\sum_{i=1}^{k} [f(\Omega_1,\ldots,\Omega_{i-1}, q_i, \Omega_{i+1},\ldots, \Omega_k) -1] \tag{2.7}$$

It should be noted that though no link fields are needed for the last record pertaining to a query, yet a termination field is needed to signal the end of the search. If the length of the termination field is denoted by ℓ_1, then the space needed by the termination field is given by:

$$k\ell_1 \tag{2.8}$$

Hence from (2.6), (2.7), and (2.8) the storage-space needed for the chaining technique is obtained as:

$$S(C) = rf(B) + \sum_{i=1}^{k} \ell f(\Omega_1,\ldots, \Omega_{i-1}, q_i, \Omega_{i+1},\ldots, \Omega_k) + k(\ell_1-\ell) \tag{2.9}$$

The formula (2.6) becomes simple in some special cases, which are given below:

<u>Special Case I:</u> <u>Frequency distribution of queries is uniform and the joint frequencies are products of individual frequencies.</u>

Such situations may occur sometimes. One example of such a situation will be when the distribution of the records with respect to all patterns of values of attributes are uniform and the query set consists of queries which specify values of an equal number of disjoint attributes. For such a situation,

$$f(\Omega_1, \Omega_2, \ldots, \Omega_{i-1}, q_i, \Omega_{i+1}, \ldots, \Omega_k) = N/k \text{ for all } i$$

$$f(\Omega_1, \ldots, q_{i_1}, \ldots, q_{i_2}, \ldots, \Omega_k) = N/k^2 \text{ for all } i_1 \text{ and } i_2 \text{ and so on.}$$

Hence
$$f(B) = N[1 - \frac{k(k-1)}{2!}\frac{1}{k^2} + \frac{k(k-1)\ (k-2)}{3!}\frac{1}{k^3} \ldots + (-1)^{k+1}\frac{1}{k^k}]$$
$$= N[1 - \frac{1}{2k}(k-1) + \frac{1}{3k^2}\binom{k-1}{2} - \frac{1}{4k^3}\binom{k-1}{3} + \ldots + (-1)^{k+1}\frac{1}{k^{k-1}}] \qquad (2.10)$$

Hence
$$S(C) = Nr[1 - \frac{1}{2k}(k-1) + \frac{1}{3k^2}\binom{k-1}{2} - \frac{1}{4k^3}\binom{k-1}{3} + \ldots + (-1)^{k+1}\frac{1}{k^{k-1}}]$$
$$+ N\ell + k(\ell_1 - \ell) \qquad (2.11)$$

<u>Special Case II:</u> <u>Invariance of proportion of pertinent records with respect to queries in any subset of the file.</u>

In such situations, the frequency distribution of the queries can be stated as

$$f(\Omega_1, \ldots, \Omega_{i-1}, q_i, \Omega_{i+1}, \ldots, \Omega_k) = Np_i \text{ where } 0<p_i<1$$

and
$$f(\Omega_1, \ldots, q_{i_1}, \ldots, q_{i_2}, \ldots, q_{i_{\ell'}}, \ldots, \Omega_k)$$
$$= N \prod_{j=1}^{\ell'} f(\Omega_1, \ldots, \Omega_{i_j-1}, q_{i_j}, \Omega_{i_j+1}, \ldots, \Omega_k)$$
$$= N \prod_{j=1}^{\ell'} p_{i_j}$$

Thus
$$f(B) = N\Big[\sum_{i=1}^{k} p_i - \sum_{i_1}^{k} \sum_{\neq\, i_2}^{k} p_{i_1}p_{i_2} + \sum^{k}\sum^{k}\sum^{k}_{i_1 \neq i_2 \neq i_3} p_{i_1}p_{i_2}p_{i_3} + \cdots$$
$$\cdots + (-1)^{k+1}\, p_1p_2\ldots,p_k\Big].$$

Hence $$S(C) = rN\left[\sum_{i=1}^{k} p_i - \sum_{i_1 \neq i_2}^{k}\sum^{k} p_{i_1}p_{i_2} + \sum_{i_1 \neq i_2 \neq i_3}^{k}\sum^{k}\sum^{k} p_{i_1}p_{i_2}p_{i_3} + \right.$$

$$\left. \cdots\cdots (-1)^{k+1} p_1 p_2 \cdots, p_k\right] + \ell\sum_{i=1}^{k} p_i + k(\ell_1 - \ell) \qquad (2.12)$$

Special Case III: Disjoint queries with respect to records.

An example of such a situation is IBM's ISAM file organization. The records belonging to the query q_i may be denoted by $\rho(q_i)$. Thus disjoint queries implies

$$\rho\left(\bigcap_{i=1}^{\ell} q_{i_j}\right) = \Phi$$

where Φ is the empty set.

Thus $f(\Omega_1, \ldots q_{i_1}, \ldots q_{i_2}, \ldots q_{i_\ell}, \ldots, \Omega_k) = 0$ for $\ell=2,3,\ldots,k$.

Hence $$f(B) = \sum_{i=1}^{k} f(\Omega_1, \ldots, \Omega_{i-1}, q_i, \Omega_{i+1}, \ldots, \Omega_k)$$

So $$S(C) = (r+\ell)\sum_{i=1}^{k} f(\Omega_1, \ldots, \Omega_{i-1}, q_i, \Omega_{i+1}, \ldots, \Omega_k) + k(\ell_1 - \ell) \qquad (2.13)$$

Special Case IV: Nested queries.

A set of queries $q_1, q_2, \ldots, q_k$ are called nested if there exists a query q_J among them for which $\rho(q_J) \supset \rho(q_i)$ for all i.

Such situations are found in many practical situations; and in many file organizations with bucket arrangements, this property of queries may be used for reducing storage space. In such a situation

$$f(B) = f(\Omega_1, \ldots, \Omega_{J-1}, q_J, \Omega_{J+1}, \ldots, \Omega_k)$$

Thus $$S(C) = rf(\Omega_1, \ldots, \Omega_{J-1}, q_J, \Omega_{J+1}, \ldots, \Omega_k)$$
$$+\sum_{i=1}^{k} \ell f(\Omega_1, \ldots, \Omega_{i-1}, q_i, \Omega_{i+1}, \ldots, \Omega_k)$$
$$+ k(\ell_1 - \ell) \qquad (2.14)$$

Special Case V: Disjoint nested queries.

In this situation, the query set is such that it can be divided into L groups (any size) which are denoted by $Q'_1, Q'_2, \ldots, Q'_L$ with the following properties:

(i) If $q_i \in Q'_i$ and $q_j \in Q'_j$ then $\rho(q_i) \cap \rho(q_j) = \phi$ for every element of Q'_i and Q'_j when $i \neq j$.

(ii) In every group Q'_i there exists a q_{J_i} such that $\rho(q_{J_i}) \supset \rho(q_{i_j})$ for every $q_{i_j} \in Q'_i$.

In such a situation

$$f(\bar{q}_1, \ldots, q_{J_1}, \ldots, q_{J_2}, \ldots, q_{J_\ell}, \ldots \bar{q}_k) = 0 \quad \text{for } \ell = 2, 3, \ldots, L.$$

Hence $$f(B) = \sum_{i=1}^{L} f(\bar{q}_1, \ldots, \bar{q}_{J_i-1}, q_{J_i}, \bar{q}_{J_i+1}, \ldots, \bar{q}_k)$$

and $$\begin{aligned} S(C) = {} & r\sum_{i=1}^{k} f(\bar{q}_1, \ldots, \bar{q}_{J_i-1}, q_{J_i}, \bar{q}_{J_i+1}, \ldots, \bar{q}_k) \\ & + \sum_{i=1}^{k} \ell f(\bar{q}_1, \ldots, \bar{q}_{i-1}, q_i, \bar{q}_{i+1}, \ldots, \bar{q}_k) \\ & + k(\ell_1 - \ell). \end{aligned} \qquad (2.15)$$

It would be interesting to compare the storage space for Inverted File Organization with that of the chaining technique. For the Inverted File Organization, the storage space is

$$S(I) = r\sum_{i=1}^{k} f(\bar{q}_1, \ldots, \bar{q}_{i-1}, q_i, \bar{q}_{i+1}, \ldots, \bar{q}_k) \qquad (2.16)$$

For comparing (2.16) with (2.9) we will assume that $\ell_1 = \ell$ because in most practical situations, the difference will be very small. Thus

$$S(I) - S(C) = r\left[\sum_{i_1 \neq i_2}^{k}\sum^{k} f(\bar{q}_1, \ldots, \bar{q}_{i_1-1}, q_{i_1}, \ldots q_{i_2}, \ldots, \bar{q}_k)\right.$$

$$-\sum^{k}\sum^{k}\sum^{k}_{i_1 \neq i_2 \neq i_3} f(Q_1, \dots, Q_{i_1-1}, q_{i_1}, \dots, q_{i_2}, q_{i_3}, Q_k) \dots$$

$$+ (-1)^{k+1} f(q_1, q_2, \dots, q_k) \Big] - \ell \sum_{i=1}^{k} f(\ _1, \dots, \ _{i-1}, q_i, \ _{i+1}, \dots, \ _k) \qquad (2.17)$$

In general, it is difficult to say whether (2.17) is positive or negative; but in special cases, some conclusions can be arrived at. If the Inverted File is constructed on addresses of records and the chaining technique is also performed on addresses, then the length of the address field may be almost equal to or less than that of a link field. When $\ell \geq r$, (2.17) is negative, hence the storage space is less for the Inverted File than the chaining technique. In many situations, the Inverted File and chaining technique are applied to the records directly. In such cases $r > \ell$, if r is large in comparison to ℓ, the storage space for the Inverted File will be greater than that for the chaining technique. The actual switching point will depend on the frequency distribution of the queries.

3. HIERARCHICAL FILES

In hierarchical files, a record contains two distinct types of segments. One part is called Header, or Master, or Level 0, or Primary segment; and the other part is composed of a number of smaller segments which are referred to as Repeated segments or subordinate segments or Level 1, 2, etc. The primary segment has a simple formatted structure with pointers or links pointing to the subordinate segments. For simplicity, it will be assumed that there is only one subordinate level which may contain repeated information, e.g., the primary segment may contain information relating to the head of a family and the subordinate level may contain formatted i formation about the members of the family.

In hierarchical files, a query may relate to primary information, or subordinate information, or a combination of both. Thus, in chaining techniques, it may be necessary sometimes to link a subset of subordinate information of one record with that of another record. Hence each unit of subordinate information must have its own identification label or key. It is needless to say, that each primary segment has an identification label. The identification of a unit of subordinate information usually contains the identification of the primary segment and some additional bits for its own identification. The interaction of a query with a record in a hierarchical file can be complex. Sometimes only the primary segment may be pertinent to a query, and other times only some units of the subordinate segment may be relevant. Thus for the chaining technique, link fields have to be attached to each unit of the segments. The length of the primary segment and the units of the subordinate units may be different, hence for calculating storage-space it is necessary to have the frequency distribution of both the primary segments and the units of the subordinate segments with respect to the queries.

The event space and the σ-field associated with it is the same as in simple-formatted file. The difference is that now a bivariate frequency function is associated with each element of the σ-field; one for the primary segments and one for the units of the subordinate segments. Thus the two frequency functions associated with the event $(\bar{q}_1, \bar{q}_2, \ldots, \bar{q}_{i-1}, \bar{q}_i, \bar{q}_{i+1}, \ldots, \bar{q}_k)$ are:

$$f_0(\bar{q}_1, \bar{q}_2, \ldots, \bar{q}_{i-1}, q_i, \bar{q}_{i+1}, \ldots \bar{q}_k) = f_{0i} \text{ (say)} \qquad (3.1)$$

and

$$f_1(\bar{q}_1, \bar{q}_2, \ldots, \bar{q}_{i-1}, q_i, \bar{q}_{i+1}, \ldots, \bar{q}_k) = f_{1i} \text{ (say)} \qquad (3.2)$$

$$i = 1, 2, \ldots, k$$

where (3.1) gives the frequency of the number of primary segments relevant to the query q_i and (3.2) gives the frequency of the number of units of subordinate segments relevant to q_i.

There can be links between primary segments of two records and also links between units of subordinate segments of two records thus frequency functions over joint events also have to be defined. They are as follows:

$$f_0(\bar{q}_1, \ldots, q_{i_1}, \ldots, q_{i_2}, \ldots, q_{i_\ell}, \ldots, \bar{q}_k) = f_{0i_1 i_2 \ldots i_\ell} \text{ (say)} \qquad (3.3)$$

and $$f_1(q_1, \dots, q_{i_1}, \dots, q_{i_2}, \dots, q_{i_\ell}, \dots, q_k) = f_{1i_1i_2\dots i_\ell}, \text{(Say)} \qquad (3.4)$$

$$\text{for } \ell = 2, 3, \dots, k.$$

Using the same type of set-theoretic calculations as in section 2, the formula for frequency of segments pertinent in a bucket when the chaining technique is used can be derived. It is given by

$$f_H(B) = \sum_{i=1}^{k} f_{0i} - \sum^{k}\sum^{k}_{i_1\neq i_2} f_{0i_1i_2} + \sum^{k}\sum^{k}\sum^{k}_{i_1\neq i_2\neq i_3} f_{0i_1i_2i_3} \dots + (-1)^{k+1} f_{0123\dots k}$$

$$+\sum_{i=1}^{k} f_{1i} - \sum^{k}\sum^{k}_{i_1\neq i_2} f_{1i_1i_2} + \sum^{k}\sum^{k}\sum^{k}_{i_1\neq i_2\neq i_3} f_{1i_1i_2i_3} \dots + (-1)^{k+1} f_{1123\dots k}.$$

(3.5)

If r_0 denotes the length of a primary segment and r_1 the length of a unit of the subordinate segment, then the space occupied by the segments in a chaining technique excluding the link fields is given by

$$S(f_H(B)) = r_0\left\{\sum_{i=1}^{k} f_{0i} - \sum^{k}\sum^{k}_{i_1\neq i_2} f_{0i_1i_2} + \sum^{k}\sum^{k}\sum^{k}_{i_1\neq i_2\neq i_3} f_{0i_1i_2i_3} \dots + (-1)^{k+1} f_{0123\dots k}\right\}$$

$$+ r_1\left\{\sum_{i=1}^{k} f_{1i} - \sum^{k}\sum^{k}_{i_1\neq i_2} f_{1i_1i_2} + \sum^{k}\sum^{k}\sum^{k}_{i_1\neq i_2\neq i_3} f_{1i_1i_2i_3} \dots + (-1)^{k+1} f_{1123\dots k}\right\}. \qquad (3.6)$$

As in section 2, if the length of the link fields are assumed to be constant and denoted by ℓ and the length of the terminating field by ℓ_1, then the storage-space for the chaining technique in hierarchical files is given by

$$S_H(C) = S(f_H(B)) + \sum_{i=0}^{l} \sum_{i=1}^{k} \ell f_{ii} + k(\ell_1 - \ell). \qquad (3.7)$$

4. RETRIEVAL TIME

The time needed to retrieve all the records pertinent to a query when chaining technique has been used can be calculated if the path of the search procedure is known. Suppose the records pertinent to the query $q_i (i=1,2,\ldots,k)$ are denoted by $r_{ij} (j=1,2,\ldots,f_i)$. The search path for q_i starts with r_{i1} and then moves to r_{i2} and then to r_{i3} and so on. The search terminates with r_{if_i}. For simplicity, it will be assumed that the records (or segments of records for hierarchical files) are of fixed length and the time needed to read a record and the link field associated with it will be denoted by τ_l.

The access time from the record r_{ij_1} to r_{ij_2}, after reading the link field associated with r_{ij_1}, will be denoted by $\tau(r_{ij_1}, r_{ij_2})$. Assuming that the link fields can be converted into access commands instantly, the time needed to retrieve all records pertinent to the query q_i is given by

$$\tau(q_i) = \sum_{j=1}^{f_i - 1} \tau(r_{ij}, r_{ij+1}) + f_i \tau_l + \tau_{oi} \qquad (4.1)$$

where τ_{oi} is the time needed to reach the first record r_{i1}.

The retrieval time for each query can be calculated from (4.1) and the total or average retrieval time for the set of k queries can be calculated from these components. In many practical situations, all queries are not used equally frequently, hence the probability or relative frequency of usage of queries have to be taken into consideration for calculating the average retrieval time for a set of queries. If the probability or the relative frequency of usage of the queries are denoted by $P_1, P_2, \ldots, P_k$, then the average retrieval time is given by

$$T = \sum_{i=1}^{k} P_i \ \tau(q_i) . \qquad (4.2)$$

In the special case, when all the queries are used equally frequently, then $P_1 = P_2 = \ldots = P_k = 1/k$ and hence

$$T = \sum_{i=1}^{k} \tau(q_i)/k .$$

In order to calculate T from (4.2) and (4.1), $\tau(r_{ij}, r_{ij+1})$'s and τ_{oi} must be known. In order to calculate these, the positions of the r_{ij}'s on the storage hardware must be known or some statistical estimates have to be considered. τ_{oi} is a special case of $\tau(r_{ij}, r_{ij+1})$ hence it is sufficient to consider calculation of $\tau(r_{ij}, r_{ij+1})$ only.

It will be assumed that the records are stored consecutively on the storage device and the chaining is in one direction, i.e., the number of records between r_{ij} and r_{ij+2} is greater than the number of records between r_{ij} and r_{ij+1}. Thus the number of records lying between r_{ij} and r_{ij+1} can vary between 0 and $\sum_{i=1}^{k} f_i - f_i$. It will also be assumed that the records are stored on a magnetic disk storage with μ records per track.

Usually statistical estimates involve less unknown measurements than exact expressions, hence a statistical estimate of $\tau(r_{ij}, r_{ij+1})$ is considered first. It will be assumed that the probability of a record being stored on a particular track is the same for all tracks and within a track the records are uniformly distributed. These assumptions are almost equivalent to uniform probability distribution of the records over the storage, except for the tracks at the beginning and end.

The number of tracks occupied by all the records is $\left[\sum_{i=1}^{k} f_i/\mu\right]+1$ where $[x]$ means the greatest integer contained in x. The first track on which r_{ij} can be stored is $[(j-1)/\mu] + 1$th track. Similarly the last track on which r_{ij+1} can be stored is

$$\left[\frac{\sum_{i=1}^{k} f_i}{\mu}\right] - \left[\frac{f_i - j - 1 - \left(\sum_{i=1}^{k} f_i - \left[\sum_{i=1}^{k} f_i/\mu\right]\mu\right)}{\mu}\right] \tag{4.3}$$

when

$$\left[\frac{f_i - j - 1 - \left(\sum_{i=1}^{k} f_i - \left[\sum_{i=1}^{k} f_i/\mu\right]\mu\right)}{\mu}\right] \geq 0 \tag{4.4}$$

When the left side of (4.4) is negative, then the last track on which r_{ij+1} can be stored is $\left[\sum_{i=1}^{k} f_i/\mu\right]+1$th track, i.e., the last track. Let

$$\nu_j = \left[\frac{\sum_{i=1}^{k} f_i}{\mu}\right] - \left[\frac{f - j - 1 - \left(\sum_{i=1}^{k} f_i - \left[\sum_{i=1}^{k} f_i/\mu\right]\mu\right)}{\mu}\right] - \left[\frac{j-1}{\mu}\right] \tag{4.5}$$

Under the assumptions already stated, the probability that the magnetic head of the disk storage has to travel x tracks to reach r_{ij+1} from r_{ij} is

$$\frac{(\nu_j - x)}{\binom{\nu_j}{2}} \qquad \text{where } x = 0,1,2,\ldots,\nu_j-1. \tag{4.6}$$

Let the seek function of the magnetic disk storage be denoted by $\tau_h(x)$ where x is the number of tracks to be travelled and $\tau_h(x)$ is the time needed measured in some suitable units. There is no explicit functional form for $\tau_h(x)$, though its value for every point has been determined by Monte Carlo methods. Thus the expected seek time for r_{ij+1} from r_{ij} is given by

$$\sum_{x=0}^{v_j-1} \frac{(v_j-x)}{\binom{v_j}{2}} \tau_h(x) \qquad (4.7)$$

Under the assumption that the records are uniformly distributed on a track and that the beginning of a search on a track is a point chosen at random, the search time of a record on a track will be half the rotational time of the disk. Thus if the rotational time of the disk is denoted by τ_r, then the average search time for a record on a track is given by

$$\frac{1}{2} \tau_r . \qquad (4.8)$$

Combining (4.7) and (4.8), a statistical estimate of $\tau(r_{ij}, r_{ij+1})$ can be obtained, which will be denoted by $\hat{\tau}(r_{ij}, r_{ij+1})$ and is given by

$$\hat{\tau}(r_{ij}, r_{ij+1}) = \sum_{x=0}^{v_j-1} \frac{(v_j-x)}{\binom{v_j}{2}} \tau_h(x) + \frac{1}{2} \tau_r . \qquad (4.9)$$

Substituting (4.9) in (4.1) an estimate of $\tau(q_i)$ can be obtained and thus an estimate of T can be obtained from (4.2).

It is interesting to compare the retrieval time of an Inverted File Organization with (4.2). In an Inverted File all the records pertinent to a query, say q_i, are stored contiguously on a disk storage. Thus the retrieval time for all records pertinent to q_i, under the assumptions already stated, is given by

$$\tau_I(q_i) = \left[\frac{f_i}{\mu}\right]\left\{\tau_h(1) + \tau_r\right\} + \left\{\frac{f_i}{\mu} - \left[\frac{f_i}{\mu}\right]\right\}\tau_r + \tau_{oi} \tag{4.10}$$

If the probability of usage of the queries are taken into consideration then the average retrieval time for Inverted File is given by

$$T_I = \sum_{i=1}^{k} P_i \, \tau_I \, (q_i) . \tag{4.11}$$

Thus a measure of the additional retrieval time that has been paid as a price for saving of storage space in chaining technique over Inverted File is given by

$$T - T_I = \sum_{i=1}^{k} P_i \, \{\tau(q_i) - \tau_I(q_i)\}$$

$$= \sum_{i=1}^{k} P_i \left\{ \sum_{j=1}^{f_i - 1} \tau(r_{ij}, r_{ij+1}) + f_i \tau_1 - \left[\frac{f_i}{\mu}\right] (\tau_h(1) + \tau_r) - \left(\frac{f_i}{\mu} - \left[\frac{f_i}{\mu}\right]\right)\tau_r \right\} \tag{4.12}$$

A statistical estimate of (4.12) can be obtained by replacing $\tau(r_{ij}, r_{ij+1})$ by its estimate given in (4.9).

<u>Exact expression for $\tau(r_{ij}, r_{ij+1})$</u>

In order to calculate an exact expression for $\tau(r_{ij}, r_{ij+1})$ the exact storage locations of r_{ij} and r_{ij+1} have to be known. In a disk storage device, which is a two dimensional storage (as the magnetic head can be switched instantly on any track on the same cylinder, the different tracks on the same cylinder are not considered as an additional dimension) the storage location of a record r_{ij}, say $S(r_{ij})$, is determined by two parameters, namely, the track number (x_{ij}) and the angular position (ω_{ij}) measured in radiants from a fixed point on the track. Thus

$$S(r_{ij}) = (x_{ij}, \omega_{ij}). \tag{4.13}$$

Thus the storage distance from the record r_{ij} to the record r_{ij+1} is given by

$$S(r_{ij}, r_{ij+1}) = (x_{ij+1} - x_{ij}, \omega_{ij+1} - \omega_{ij}) \tag{4.14}$$

$\tau(r_{ij}, r_{ij+1})$ contains two components, the access (seek) time and the search time on the track. The access time is given by

$$\tau_h(x_{ij+1} - x_{ij}). \tag{4.15}$$

For calculating the search time, two functions have to be introduced. Let

$f_c\ (x)$ = the fractional part of x

$\tau_s\ (\omega_1, \omega_2)$ = the time needed for the disk to rotate from the angular position ω_1 to ω_2.

As the disk rotates 2π radians in time τ_r, hence the search time is

$$\tau_s\left(\frac{2\pi}{\tau_r}\ f_c\left\{\frac{\tau_h(x_{ij+1}-x_{ij})}{\tau_r}\right\} + \omega_{ij}, \omega_{ij+1}\right). \tag{4.16}$$

Hence,

$$\tau(r_{ij}, r_{ij+1}) = \tau_h(X_{ij+1} - X_{ij}) + \tau_s\left(\frac{2\pi}{\tau_r} f_c \left\{\frac{\tau_h(X_{ij+1} - X_{ij})}{\tau_r}\right\} + \omega_{ij}, \omega_{ij+1}\right). \tag{4.17}$$

SUMMARY

It has been shown that the problem of file organization can be looked at as a problem in time and space. Retrieval time and storage space are the two important factors. Both of these factors cannot be reduced simultaneously. They move in opposite directions. A file organization technique attempts to balance these two factors. In this paper, these concepts have been brought to light using "chaining technique" of file organization. Mathematical formulae for retrieval time and storage space need have been calculated. It has been shown that some of the formulas become simple when statistical assumptions are made. Retrieval time and storage space required in chaining technique have been compared with that of Inverted File organization. One important aspect which the author would like to emphasize is that it is difficult to combine retrieval time and storage space requirement into one measure of goodness, and this should not be attempted when comparing different file organization techniques.

ACKNOWLEDGEMENT

The author would like to thank Dr. Michael E. Senko and Dr. Frank Palermo of the Information Sciences Department of IBM Research Laboratory for their valuable help rendered during the work.

REFERENCES

1. Abraham, C. T., Ghosh, S. P., and Ray-Chaudhuri, D. K., (1968), "File Organization Schemes Based on Finite Geometries," Information and Control 12, pp. 143-163.

2. Baker, F. T., (1963), "Some Storage Organization for Use With Disk Files." IBM Federal System Division Report.

3. Buchholz, W., (1963), File Organization and Addressing. IBM Systems Journal 2 pp. 86-111.

4. Davis, D. R. and Lin, A. D., (1965), Secondary Key Retrieval Using an IBM 7090-1301 System. Commun. A.C.M., 8, pp. 243-246.

5. Gray, H. J., Landauer, W. I., Lefkowitz, D., Litwin, S., and Prywes, N. S., (1961), The Multiple List System, University of Pennsylvania Report.

6. IBM Report, (1967), Formatted File Organization Techniques, Contract A.F. 30(602)-4088.

SECTION IV

ORGANIZATION OF RECORDS WITH UNEQUAL MULTIPLE-VALUED ATTRIBUTES AND COMBINATORIAL QUERIES OF ORDER 2

S. P. Ghosh

ORGANIZATION OF RECORDS WITH UNEQUAL MULTIPLE-VALUED ATTRIBUTES AND COMBINATORIAL QUERIES OF ORDER 2*

S. P. Ghosh

Information Sciences Department
Research Division
San Jose, California

ABSTRACT: This paper develops theories for constructing filing schemes for formatted files with unequal-valued attributes when the query set contains all queries which specify two values from two attributes. These filing schemes provide a set of buckets for storing accession numbers of records. The retrieval rule is based on identifying a bucket from a query by solving algebraic equations over finite fields. The theories underlying these filing schemes are based on properties of deleted finite geometries. Expressions for retrieval time and storage redundancy are also given.

1. INTRODUCTION

A large volume of data may be stored in many ways in the storage area of a computer. Usually, these are stored in small blocks called records. A record is an information block and can have any structure, but we will discuss the case where this block of information is represented by an n-vector, each component being a number providing information regarding one of a set of k attributes $A_1, A, \ldots, A$. These numbers are called values of the attributes. Each record has a record-identification and is denc ed by i. Thus, if v_{ij} denotes the value of the j^{th} attribute of the i^{th} record, then the structure of the record is

$$f(i) = (i; v_{i1}, v_{i2}, \ldots, v_{i})$$

The collection of records is called a file. One of the main purposes of storing records in a computer is the ability to recall any subset of the file which meets certain criteria. This criteria is also called a "query." A query may specify a key, or a collection of keys, or a particular value of an attribute or a collection of values of different attributes or a combination of keys and values of attributes. In general, a query may be stated by the user in many different forms but at the particular stage of retrieval process, when the query has to interact with the file, it has similar structures to the ones described above. In this paper, we confine ourselves to structures of queries which specify a collection of values of attributes. Thus, we may want to retrieve all records for which the attributes $A_{j_1}, A_{j_2}, \ldots, A_{j_g}$ have the values $v_{j_1}, v_{j_2}, \ldots, v_{j_g}$ $(g \leq k)$, whatever be the values of the other $k - g$ attributes. We shall denote this query by

$$Q \begin{pmatrix} A_{j_1}, A_{j_2}, \ldots, A_{j_g} \\ \\ v_{j_1}, v_{j_2}, \ldots, v_{j_g} \end{pmatrix}$$

All records for which the attributes A_{j_i} have the value v_{j_i} for $i = 1, 2, \ldots, g$ are said to satisfy the query or pertain to the query.

Records within a file are organized according to a scheme to reduce the time needed to retrieve pertinent records for a given class of queries. The problem of file organization is fairly simple when queries relate to only one attribute. A summary of this type of work has been given by Buchholz (1963). Prywes et al. (1961) attempted to resolve the problem of minimizing search time for multiple attribute queries by grouping attributes into composite attributes and forming a tree structure, and Davis and Lin (1965) suggested the formation of partition classes by considering possible values of logical fields. Abraham, Ghosh and Ray-Chaudhuri (1968) used finite geometry to construct combinatorial filing schemes for binary attributes. Their method consisted of forming groups of records in such a manner that the group containing records pertaining to a given query could be determined algebraically, thus, expediting the search. Ray-Chaudhuri (1968) discussed some further combinatorial properties of file organization schemes for binary valued attributes. Ghosh and Abraham (1968) developed the theory for file organization schemes for multiple valued attributes where attributes have an equal number of possible values and the queries specify two values of two attributes. This result was generalized by Abraham, Bose and Ghosh (1967) to the case where the queries specify t ([illegible]) values of t attributes. In this paper, properties of t-independent linear forms over a Galois field GF(s) were used for constructing file organization schemes. In a subsequent preliminary report, Abraham, Bose and Ghosh (1967) showed that by using some dependent linear forms along with independent linear forms, filing schemes can be constructed when attributes have unequal values which are some multiples of s, where s is a power of a prime number. In this paper it will be shown that when the query set consists of all queries which specify two values from two attributes, then it is possible to develop combinatorial filing schemes if the attributes have unequal values. These results will be developed in two parts. In the first part, deleted (or full) finite geometries will be used to develop filing schemes when the number of values the attributes can take are any multiples of s. In the second part, partially deleted geometries will be used to construct filing schemes when the attributes can take any number of values.

2. UNEQUAL VALUED FILING SCHEMES

In most computerized filing systems, the records are stored in some relatively slow storage area. The starting address of a segment of the storage device, where the record is stored in its entirety, is called the accession number of the record. A set of comparatively faster memory or storage device (like the cylinder of a random access disc package) is reserved for storing the accession numbers. Let this set be denoted by S. In this paper, file organization schemes are rules for storing accession numbers of records in S, and retrieving the pertinent accession numbers when a query is asked. These rules are referred to as storage rule and retrieval rule. In all filing schemes which are in practice today, the retrieval rule consists of matching operation, whereas in the filing schemes discussed here, the retrieval rule can be algebraic computation only, by proper choice of hardware. In order to achieve this property, the storage rule is developed from structures of some finite geometries. The query set consists of queries which specify two values of two attributes, thus, when $\ell > 2$, an accession number of a record is stored in more than one location in S. The redundant storage of accession numbers can be avoided by using complex chaining techniques and, thus, increasing search time. In the storage rule which is discussed in this paper, a limited amount of chaining is used to reduce the redundant storage of accession numbers but not increase the search time too much. Additional storage requirement and search time formulae for filing schemes are discussed in detail in latter sections.

A collection of all queries which specify t values of t different attributes are defined to be combinatorial query sets of order t, and will be denoted by Q_t.

A Multiple-valued Filing Scheme with parameters $(t, n_1, n_2, \ldots, n_\ell, b)$ is defined to be an arrangement of the accession numbers of records with ℓ attributes, where the vector of the number of values these attributes can take is given by $(n_1, n_2, \ldots, n_\ell)$, in b groups (buckets), which are not necessarily mutually exclusive and which satisfy the following properties:

(1) The accession numbers in a bucket is a subset of all accession numbers (property of redundancy).

(2) Associated with each bucket is an algebraic identifier. There is a correspondence between the algebraic identifier

of a bucket and the accession numbers in the bucket (property of identifiability).

(3) Corresponding to any query which specifies t ($t \geq 2$) values of t different attributes there exists a unique bucket (property of uniqueness).

A multiple valued filing scheme corresponding to a combinatorial query set of order t will be denoted by $MVFS_t$.

Ghosh and Abraham (1968) have constructed Balanced Multiple-valued Filing Schemes of order 2. In those filing schemes $n_1 = n_2 = \ldots, = n_\ell = s$ and, thus, it was possible to have each bucket containing the accession numbers of an equal number of queries. In $MVFS_t$ when the n_i's are not equal, in general, the buckets may not contain the accession numbers of an equal number of queries. In some of the cases discussed in this paper, $MVFS_2$ will be balanced with respect to queries.

The problem of construction of $MVFS_t$ can be considered as a problem in combinatorial algebra which may be stated as follows: given ℓ sets of sizes n_1, n_2, ..., n_ℓ how can b groups be formed such that any subset of t elements from t (of the ℓ) sets will be contained in one and only one group, and it should be possible to determine that group algebraically from the subsets. No satisfactory mathematical theories are known which will provide a direct answer to this problem so an attempt has been made to take finite geometries, which have symmetric structures, and delete some portions to provide an answer to this problem.

3. DELETED FINITE GEOMETRIES

An N dimensional finite Euclidean geometry defined over a Galois Field GF(s) where $s = p^n$ and p is a prime integer is denoted by EG(N, s). Points in this geometry are denoted by n-tuples of the form $x = (x_1, x_2, \ldots, x_N)$ where $x_i \in GF(s)$. A t-dimensional flat in this geometry is defined by a set of points which satisfy N-t independent linear equations with coefficients in GF(s). There are s^N points in this geometry, and any t-dimensional flat contains s^t points. The number of t-flats in EG(N, s) is equal to $s^{N-t}\phi(N-1, t-1, s)$ where

$$\phi(N, t, s) = \frac{(s^{N+1}-1)(s^N-1)\ \ldots\ (s^{N-t+1}-1)}{(s^{t+1}-1)(s^t-1)\ \ldots\ (s-1)}$$

When some structures from a finite geometry are deleted, then the resulting geometry is called a deleted geometry, e.g. some lines of a EG(N, s) may be deleted. The remaining lines and all the points of the geometry may be used to construct a filing scheme. This technique was used by Ghosh and Abraham (1968) to construct balanced multiple-valued filing schemes of order 2. When some points of the geometry are deleted, then irregular structures are obtained, i.e., all the t-flats of the geometry do not have the same number of points. These types of deleted irregular geometries will be called partially deleted geometries.

Example 3.1

Consider a EG(2, 3). There are 9 points in this geometry which may be represented (without a comma separation between the coordinates) as 00, 01, 02, 10, 11, 12, 20, 21, 22. The 12 lines of the geometry with their algebraic equations are:

Equation	Points
$x_1 = 0$	00, 01, 02
$x_1 = 1$	10, 11, 12
$x_1 = 2$	20, 21, 22
$x_2 = 0$	00, 10, 20
$x_2 = 1$	01, 11, 21
$x_2 = 2$	02, 12, 22
$x_1+x_2 = 0$	00, 12, 21
$x_1+x_2 = 1$	01, 10, 22
$x_1+x_2 = 2$	02, 20, 11
$x_1+2x_2 = 0$	00, 11, 22
$x_1+2x_2 = 1$	10, 21, 02
$x_1+2x_2 = 2$	20, 12, 01

If the points 00, 12, and 21 are deleted from this geometry, then we get a partially deleted geometry whose lines are:

$x_1 = 0$	01, 02	$x_2 = 0$	10, 20
$x_1 = 1$	10, 11	$x_2 = 1$	01, 11
$x_1 = 2$	20, 22	$x_2 = 2$	20, 22
$x_1+x_2 = 1$	01, 10, 22	$x_1+2x_2 = 0$	11, 22
$x_1+x_2 = 2$	02, 20, 11	$x_1+2x_2 = 1$	10, 02
		$x_1+2x_2 = 2$	20, 01

This partially deleted geometry contains 9 points and 11 lines. 9 of the lines contain 2 points each, and 2 lines contain 3 points each.

If only the 3 lines: $x_1 = 0$, $x_1 = 1$ and $x_1 = 2$ are deleted from EG(2, 3), then the resulting geometry is a deleted geometry with 9 lines and each of the lines contains 3 points.

A <u>spread</u> in a finite geometry is a collection of disjoint flats whose union covers the geometry. In the example 3.1, the 3 lines $x_1+x_2 = 0$, $x_1+x_2 = 1$, and $x_1+x_2 = 2$ form a spread. Each of these three lines are called an <u>element of the spread</u>.

4. MULTIPLE VALUED FILING SCHEMES AND DELETED GEOMETRIES

Consider an EG(N, s) and a point in this geometry denoted by $(x_1, x_2, \ldots, x_N)$ where $x_i \in GF(s)$. An N-1 dimensional flat in this geometry is defined by the set of s^{N-1} points which satisfy the following equation $a_{11} x_1 + a_{12} x_2 + \ldots + a_{1N} x_N = c_1$ where the a_{1j}'s and c_1 are elements of GF(s). If a_{1j}'s are kept fixed and c_1 is given the s different values of GF(s), then we get s (N-1)-flats which form a spread. Suppose these s (N-1)-flats are identified with s attributes and the s^{N-1} points on each of the elements of the spread are identified with s^{N-1} values which the attributes can take. Thus, a 1-1 correspondence between a formatted file with s attributes where each attribute can take s^{N-1} values is established. The file can have $s^{s(N-1)}$ distinct records and each record is identified with an s-tuple of points in the geometry. There are $s^{N-1}\,\phi(N-1, 0, s) = s^{N-1}(s^N-1)/(s-1)$ lines in the geometry and each element of the spread $a_{11}x_1 + a_{12}x_2 + \ldots + a_{1N}x_N = c_1$ $(c_1 \in GF(s))$ contains $s^{N-2}\,\phi(N-2, 0, s) = s^{N-2}(s^{N-1}-1)/(s-1)$ lines. If all the lines which lie completely on the elements of the spread are deleted then we will get a deleted geometry with $s^{N-1}(s^N-1)/(s-1) - s^{N-1}(s^{N-1}-1)/(s-1) = s^{2(N-1)}$ lines. The buckets of the filing scheme will be identified with the $s^{2(N-1)}$ lines of the deleted geometry. Each line in EG(N, s) is represented by a system of N-1 linearly independent equations. Thus, the matrix of the coefficient of the equations, which have order (N-1) x (N+1) may be used as a bucket identification. The storage rule for the record $f(i) = (i;\ v_{i1}, v_{i2}, \ldots, v_{is})$ is defined as follows: the accession number of the record f(i) is stored in all buckets which have at least two points which are common with the s-tuple point-representation of f(i). Inside the bucket the accession numbers are subdivided into $s(s-1)/2$ sub-buckets corresponding to the $s(s-1)/2$ pairs of points of the bucket. An accession number of a record may be entitled to be stored in more than one sub-bucket within a bucket but in order to reduce redundant storage, an accession number will be stored only in one sub-bucket within a bucket and properly chained (chaining technique) with the other relevant sub-buckets within the bucket. Chaining between buckets is avoided to reduce complications. The same rule is applied to store the accession numbers of all the records in the file.

Example 4.1

Consider a EG(3, 3) and a spread of planes in this geometry. Suppose the spread is represented by the equations $x_1 = 0$, $x_1 = 1$, and $x_1 = 2$. We use this geometry and this spread to construct a filing scheme for a file with 3 attributes, where each attribute can take 9 values. The 3 attributes A_0, A_1, and A_2 will correspond to the 3 elements of the spread and the values that these attributes can take will correspond to the points on the spread. Let v'_{ij} be the j^{th} value of the i^{th} attribute ($j = 0, 1, \ldots, 8$, $i = 0, 1, 2$), and the correspondence between the values and the points are as follows: the point (ijk) will correspond to the value v'_{im} where $m = 3j+k$. The buckets will correspond to the 81 lines of the geometry which do not lie completely in any one of the three planes $x_1 = 0$ or $x_1 = 1$ or $x_1 = 2$. Suppose a record is composed of the values v'_{03}, v'_{10} and v'_{24}, i.e., $f(i) = (i; v'_{03}, v'_{10}, v'_{24})$. Then the point-representation of this record is $f(i) = (i; 010, 100, 211)$. According to the storing rules, the accession number of this record is stored in three buckets corresponding to the three lines: $x_3 = 0$, $x_1+x_2 = 1$; $x_2 = 1$, $x_1+x_3 = 1$ and $x_1+2x_2 = 1$, $x_1+x_2+x_3 = 1$. These three buckets may be identified by their matrix of coefficients as

$$\begin{pmatrix} 0010 \\ 1101 \end{pmatrix}, \quad \begin{pmatrix} 0101 \\ 1011 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 1201 \\ 1111 \end{pmatrix}$$

Within the bucket with label $\begin{pmatrix} 0010 \\ 1101 \end{pmatrix}$ the accession number of the record f(i) will be stored in the sub-bucket corresponding to the pair 010, 100. This sub-bucket may be given a sub-bucket label 010100 (obtained by concatenating the ordered pair).

The storage rule defined above, will provide a filing scheme which answers all queries of a combinatorial query set of order 2, i.e., Q_2 or

$$Q \begin{pmatrix} A_{j_1} & A_{j_2} \\ v'_{j_1 j} & v'_{j_2 k} \end{pmatrix}$$

The retrieval rule for any query belonging to Q_2 will consist of five steps:

(i) The specified values of the attributes will be converted into their point-representation.

(ii) The equation of the line containing the particular pair of points is determined by solving N-dimensional algebraic equations over GF(s).

(iii) The bucket corresponding to the line is reached by appropriate command to the storage unit.

(iv) The appropriate sub-bucket is reached by matching the sub-bucket labels with the query within the bucket.

(v) The accession numbers are retrieved from the sub-bucket and the corresponding records are retrieved from the storage.

In a EG(N, s) through any two points there passes only one line, thus, step (ii) of the retrieval rule will always provide a unique bucket, and hence, the property of uniqueness of $MVFS_2$ will be satisfied. In the preceding discussions, it has also been established that the above filing scheme satisfies the property of redundancy and identifiability. Thus, we have the following theorem:

Theorem 4.1

There exists a $MVFS_2$ with parameters t = 2, [illegible] = s, $n_1 = n_2 = \ldots, = n_s = s^{N-1}$ and $b = s^{2(N-1)}$, where s is the power of a prime number.

Let the elements of GF(s) be denoted by $\alpha_0, \alpha_1, \ldots, \alpha_{s-1}$. Consider a EG(N, s) and choose an (N-1) dimensional spread in it. Let it be denoted by $a_{11}x_1+a_{12}x_2+ \dots +a_{1N}x_N = c_1$ where the a_{1j}'s are fixed and c_1 varies over all the s elements of GF(s).

Consider $s_1 < s$ elements of this spread which are represented by

$$\begin{aligned}
&a_{11}x_1+a_{12}x_2+ \dots +a_{1N}x_N = \alpha_0 \\
&a_{11}x_1+a_{12}x_2+ \dots +a_{1N}x_N = \alpha_1 \\
&\vdots \\
&a_{11}x_1+a_{12}x_2+ \dots +a_{1N}x_N = \alpha_{s_1}
\end{aligned} \qquad (4.1)$$

and associate with them s_1 attributes with s^{N-1} values each. The s^{N-1} points on each of these (N-1) dimensional flats represents the values of the attributes.

Consider another s_2, where $s_1+s_2 < s$, elements of the (N-1) dimensional spread and partition each (N-1)-flat into (N-2) dimensional spreads. These may be represented as follows:

$$\begin{aligned}
&a_{11}x_1+a_{12}x_2+ \dots +a_{1N}x_N = \alpha_{s_1+j} \\
&a_{21}x_1+a_{22}x_2+ \dots +a_{2N}x_N = c_2
\end{aligned} \qquad (4.2)$$

where the a_{ik}'s are fixed and c_2 varies over all the s elements of GF(s) and j varies over 1 to s_2.

In (4.2) the collection of s (N-2)-flats for a fixed value of j is a (N-2) dimensional spread of a (N-1)-dimensional flat. Associate with these $s_2 s$ (N-2) flats, $s_2 s$ attributes with s^{N-2} values each. The values of the attributes are associated with the points of the (N-2)-flats. This technique can be repeated (N-1) times and, thus, establishing an association between a EG(N, s) and a file with the following type of structure:

$$\begin{array}{ll} s_1 & \text{attributes have } s^{N-1} \text{ values each} \\ s_2 s & \text{attributes have } s^{N-2} \text{ values each} \\ s_3 s^2 & \text{attributes have } s^{N-3} \text{ values each} \\ \cdot & \\ \cdot & \\ \cdot & \\ s_{N-1} s^{N-2} & \text{attributes have } s \text{ values each} \\ & \text{where } s_1+s_2+ \dots +s_{N-1} = s \end{array} \tag{4.3}$$

In this association, the $s_{i+1} s^i$ attributes with s^{N-i-1} values each will be associated with the following (N-i-1)-flats

$$\begin{array}{l} a_{11}x_1+a_{12}x_2+ \dots + a_{1N}x_N = \alpha_{j_1} \\ a_{21}x_1+a_{22}x_2+ \dots +a_{2N}x_N = c_2 \\ \cdot \\ \cdot \\ \cdot \\ a_{i+1\,1}x_1+a_{i+1\,2}x_2+ \dots +a_{i+1\,N}x_N = c_{i+1} \end{array} \tag{4.4}$$

where $a_{k\ell}$'s are fixed, j_1 varies between $s_1+ \dots +s_i+1$

and $s_1 + \ldots + s_{i+1}$, and $c_2, c_3, \ldots, c_{i+1}$ vary over all elements of GF(s).

A deleted geometry will be constructed from EG(N, s) by deleting all lines which lie completely on every flat which is associated with an attribute, i.e., all the $s_1 s^{N-2}\,\phi(N-2, 0, s)$ lines which lie completely on the s_1 (N-1)-flats given in (4.1) will be deleted; the $s_2 s^{N-2}\,\phi(N-3, 0, s)$ lines which lie completely on the $s_2 s$ (N-2)-flats defined in (4.2) will be deleted and so on. Thus,

$$s_1 s^{N-2}\,\phi(N-2,0,s) + s_2 s^{N-2}\,\phi(N-3,0,s) + s_3 s^{N-2}\,\phi(N-4,0,s) + \ldots$$
$$\ldots + s_{N-2} s^{N-2}\,\phi(1,0,s) + s_{N-1}$$

lines will be deleted from EG(N, s). This deleted geometry will have

$$s^{N-2}\,(s\,\phi(N-1,0,s) - s_1\,\phi(N-2,0,s) - \ldots - s_{N-2}\,\phi(1,0,s) - s_{N-1}) \quad (4.5)$$

lines. Each of these lines will correspond to a bucket in the filing scheme. The storage rule and the retrieval rule for the accession numbers of the records will be the same as for the MFS_2 with parameters $t = 2$, $\ell = s$, $n_1 = n_2 = \ldots = n_s = s^{N-1}$, $b = s^{2(N-1)}$. If a query specifies two values of two attributes, then the line passing through the point-representation of the query is contained in the deleted geometry described above. If the query represents two values of the same attribute, then the line passing through the point-representation of query is not contained in the deleted geometry and, hence, there will be no bucket corresponding to that query. Thus, we have the following theorem:

Theorem 4.2

There exists a $MVFS_2$ with parameters $t = 2$, $\ell = s_1 + s_2 s + s_3 s^2 + \ldots + s_{N-1} s^{N-2}$ where $s_i \geq 0$ for all i and $s_1 + s_2 + \ldots + s_{N-1} = s$, s being the power of a prime number,

$$n_1 = n_2 = \ldots = n_{s_1} = s^{N-1}$$

$$n_{s_1+1} = n_{s_1+2} = \ldots = n_{s_1+s_2 s} = s^{N-2}$$

$$\vdots$$

$$n_{\ell - s_{N-1} s^{N-2} + 1} = \ldots = n_{\ell} = s$$

and $b = s^{N-2} \{ s\, \phi(N-1,0,s) - s_1\, \phi(N-2,0,s) - \ldots - s_{N-2}\, \phi(1,0,s) - s_{N-1} \}$.

Example 4.2

Consider a file which has two attributes with 9 values each and three attributes with 3 values each. The attributes are denoted by A_0, A_1, A_2, A_3 and A_4. The jth values of the ith attribute is represented by v'_{ij} ($j = 0,1, \ldots, 8$ for $i = 0,1$, and $j = 0,1,2$ for $i = 2,3,4$). For constructing a MVIS$_2$ for this file, consider a EG(3,3). The three planes $x_1 = 0$, $x_1 = 1$, and $x_1 = 2$ form a spread in EG(3,3). Associate the 9 points of $x_1 = 0$ with the 9 values of A_0 and the 9 points of $x_1 = 1$ with the 9 values of A_1, i.e., $v'_{0m} \approx (0jk)$, $v'_{1m} \approx (1jk)$ where $m = 3j+k$; $j,k = 0,1,2$. The three lines $x_1 = 2$, $x_2 = 0$; $x_1 = 2$, $x_2 = 1$; and $x_1 = 2$, $x_2 = 2$ form a spread on the plane $x_1 = 2$. Associate these three lines with the three attributes A_2, A_3 and A_4 and the points on the lines with the values of the attributes in the following manner:

$$v'_{2j} \approx (20j),\ v'_{3j} \approx (21j),\ v'_{4j} \approx (22j),\quad j = 0,1,2.$$

EG(3,3) contains 117 lines from which the following 27 lines are deleted:

$x_1 = 0,\ x_2 = 0;$	$x_1 = 0,\ x_2 = 1;$	$x_1 = 0,\ x_2 = 2$
$x_1 = 0,\ x_3 = 0;$	$x_1 = 0,\ x_3 = 1;$	$x_1 = 0,\ x_3 = 2$
$x_1 = 0,\ x_2+x_3 = 0;$	$x_1 = 0,\ x_2+x_3 = 1;$	$x_1 = 0,\ x_2+x_3 = 2$
$x_1 = 0,\ x_2+2x_3 = 0;$	$x_1 = 0,\ x_2+2x_3 = 1;$	$x_1 = 0,\ x_2+2x_3 = 2$
$x_1 = 1,\ x_2 = 0;$	$x_1 = 1,\ x_2 = 1;$	$x_1 = 1,\ x_2 = 2$
$x_1 = 1,\ x_3 = 0;$	$x_1 = 1,\ x_3 = 1;$	$x_1 = 1,\ x_3 = 2$
$x_1 = 1,\ x_2+x_3 = 0;$	$x_1 = 1,\ x_2+x_3 = 1;$	$x_1 = 1,\ x_2+x_3 = 2$
$x_1 = 1,\ x_2+2x_3 = 0;$	$x_1 = 1,\ x_2+2x_3 = 1;$	$x_1 = 1,\ x_2+2x_3 = 2$

$x_1 = 2, x_2 = 0$; $x_1 = 2, x_2 = 1$; $x_1 = 2, x_2 = 2$.

Thus, a deleted EG(3,3) with 90 lines is obtained. The buckets will correspond to these 90 lines. If the accession numbers of the records are stored in these 90 buckets following the same storage rule as in example 4.1, then we will get a $MVFS_2$ with parameters $t = 2$, $\ell = 5$, $n_1 = n_2 = 9$, $n_3 = n_4 = n_5 = 3$ and $b = 90$.

5. REDUNDANCY OF MULTIPLE VALUED FILING SCHEMES

In the filing schemes discussed in the previous section, an accession number of a record is stored in more than one bucket, and this will give rise to redundant storage. The average number of times the accession numbers are stored in the filing scheme is called the redundancy of the filing scheme. The redundancy of a filing scheme will depend on the following properties:

(i) The frequency distribution of the different types of records in the file;

(ii) The patterns of values of the attributes in the different buckets;

(iii) The number of different values the different attributes can take;

(iv) The types of queries in the query set.

In deriving the algebraic expressions for the redundancy, it is assumed that the query set is of the type Q_2 and the frequency distribution of the different types of records in the file is uniform. Thus, the number of records in the file is some multiple of $\prod_{i=1}^{\ell} n_i$. As we are calculating the redundancy, it is sufficient to assume that the total number of records is $\prod_{i=1}^{\ell} n_i$. Suppose j^{th} bucket in the filing scheme has the following type of structure:

$$\{A_{j_1} = v_{j_1 \ell_1}, A_{j_2} = v_{j_2 \ell_2}, \ldots A_{j_{k_j}} = v_{j_{k_j} \ell_{k_j}}\}$$

Let us denote by S_j the set of attributes which are represented in the j^{th} bucket, i.e.,

$$S_j = \{A_{j_1}, A_{j_2}, \ldots, A_{j_{k_j}}\} \qquad (5.1)$$

$S_j - A_i$ will denote the set of attributes of S_j from which A_i has been deleted. Then under the assumption of uniform distribution of records, it is easy to see that the number of accession numbers which will not be stored in the j^{th} bucket is given by:

$$\prod_{i \in S_j} (n_{j_i} - 1) \prod_{i \in \overline{S}_j} n_i + \prod_{i \in \overline{S}_j} n_1 \cdot \sum_{i \in S_j} \prod_{i' \in S_j - A_i} (n_{j_{i'}} - 1)$$

where $\overline{S}_j$ is the complement of the set S_j

$$= \prod_{i \in \bar{S}_j} n_i \left[\prod_{i \in S_j} (n_{j_i} - 1) + \sum_{i \in S_j} \prod_{i' \in S_j - A_i} (n_{j_{i'}} - 1) \right]$$

$$= \prod_{i \in \bar{S}_j} n_i \prod_{i \in S_j} (n_{j_i} - 1) \left[1 + \sum_{i \in S_j} 1/(n_{j_i} - 1) \right] \qquad (5.2)$$

Thus, the total number of accession numbers in the jth bucket is

$$\prod_{i=1}^{\ell} n_i - \prod_{i \in \bar{S}_j} n_i \prod_{i \in \bar{S}_j} (n_{j_i} - 1) \left[1 + \sum_{i \in S_j} 1/(n_{j_i} - 1) \right]$$

$$= \prod_{i=1}^{\ell} n_i \left\{ 1 - \prod_{i \in S_j} \left(\frac{n_{j_i} - 1}{n_{j_i}} \right) \left[1 + \sum_{i \in S_j} 1/(n_{j_i} - 1) \right] \right\} \qquad (5.3)$$

The total number of records in the filing scheme is:

$$\prod_{i=1}^{\ell} n_i \sum_{j=1}^{b} \left\{ 1 - \prod_{i \in S_j} \left(\frac{n_{j_i} - 1}{n_{j_i}} \right) \left[1 + \sum_{i \in S_j} 1/(n_{j_i} - 1) \right] \right\}$$

Thus, the redundancy of the filing scheme is given by:

$$R = \sum_{j=1}^{b} \left\{ 1 - \prod_{i \in S_j} \left(\frac{n_{j_i} - 1}{n_{j_i}} \right) \left[1 + \sum_{i \in S_j} 1/(n_{j_i} - 1) \right] \right\} \qquad (5.4)$$

Consider the $MVFS_2$ with parameters $\ell = s$,

$n_1 = n_2 = \ldots = n_s = s^{N-1}$, $b = s^{2(N-1)}$. If a file has uniform distribution of records, then the number of accession numbers is $s^{s(N-1)}$. Here $S_j = \{A_1, A_2, \ldots, A_s\}$ for all j and $\overline{S}_j$ = the empty set. Thus, from (5.4), the redundancy is given by

$$R = s^{2(N-1)} \cdot \left(1 - \frac{s^{N-1}-1}{s^{N-1}}\right)^{s} \left(1 + \frac{s}{s^{N-1}-1}\right) \qquad (5.5)$$

In the filing scheme discussed in example 4.1, $N = 3$ and $s = 5$, thus, the redundancy of that filing scheme is 2.778.

Consider the $MVFS_2$ with parameters $s = s_1 + s_2 s + s_3 s^2 + \ldots$ $\ldots + s_{N-1} s^{N-2}$ and the attributes divided into N-1 groups $(G_1, G_2, \ldots, G_{N-1})$ where the ith group G_i contains $s_i s^{i-1}$ attributes each having s^{N-i} values, $i = 1, 2, \ldots, N-1$. It has been shown in theorem 4.2, that $b = s^{N-2}\{s\,\Phi(N-1,0,s)$ $- s_1\,\Phi(N-2,0,s) \ldots - s_{N-2}\,\Phi(1,0,s) - s_{N-1}\}$. When the records are uniformly distributed then the total number of different types of records is $s^{\sum_{i=1}^{N} s_i s^{i-1}(N-i)}$. Here

$S_j = \{A_1, A_2, \ldots, A_{s_1}, s_2\ A_i$'s from G_2, $s_3\ A_i$'s from G_3, $\ldots$

$\ldots, s_{N-1}\ A_i$'s from $G_{N-1}\}$ for all j.

$\overline{S}_j = \{s_2(s-1)\ A_i\text{'s from } G_2,\ s_3(s^2-1)\ A_i\text{'s from } G_3, \ldots$

$\ldots s_{N-1}(s^{N-2}-1)\ A_i\text{'s from } G_{N-1}\}$

Substituting in (5.4) the redundancy of this MVFS$_2$ is given by

$$R = s^{N-2}\ \{s\ \phi(N-1,0,s) - \sum_{i=1}^{N-1} s_i\ \phi(N-i-1,0,s)\}\left\{1 - \prod_{i=1}^{N-1}\left(\frac{s^{N-i}-1}{s^{N-i}}\right)^{s_i}\right.$$

$$\left.\times\left[1 + \sum_{i=1}^{N-1}\frac{s_i}{s^{N-i}-1}\right]\right\} \qquad (5.6)$$

In the filing scheme discussed in example 4.2, $N = 3$, $s = 3$, $k = 5$, $n_1 = n_2 = 9$, $n_3 = n_4 = n_5 = 3$, $s_1 = 2$, $s_2 = 1$, $b = 90$. The number of different types of records in that file under uniform distribution is 2187. The redundancy of that filing scheme is 7.037.

6. RETRIEVAL TIME

In the filing schemes discussed, the retrieval time is composed of two components:

(i) Retrieval of the accession numbers of the relevant documents;

(ii) Retrieval of the records when accession numbers are given.

The time needed for these two components is denoted by T_1 and T_2. T_1 and T_2 depend on the structure of the file, the lengths of the records, the filing scheme, the hardware system and what operation of the retrieval procedure is carried out by which component of the computer system. It will be assumed that the main file of records are stored in a random access disc pack and also another disc pack is available for construction of the filing scheme.

The attributes and the values which they can assume have a representation in the computer. The first step in the retrieval procedure consists of converting the query representation to its point representation. This may be achieved by a simple table look-up. Let

t_1 = time needed for converting a query to its point representation.

The points are used to determine the algebraic equation and, hence, the bucket label pertinent to the query. This operation is carried out in the central processing unit. Let

t_2 = time needed to solve the algebraic equations to determine the bucket label.

It is easy to see that the coefficients of the algebraic equations (properly concatenated) of all the lines of a finite geometry form sets of linearly ordered consecutive elements. In MFS_2, although all the lines of the finite geometry do not correspond to buckets, the coefficients of the algebraic equations of the lines, which correspond to buckets, can be made to correspond to a set of linearly ordered consecutive elements because the deleted lines have a systematic pattern. Thus, the buckets can be made to correspond with consecutive tracks on a disc. Different practical situations may necessitate corresponding one bucket to more than one track, or fraction of a track, but all these can be taken care of by simple mathematical transformation. Thus, in the retrieval procedure, the computer, after solving athe algebraic equation, can give a direct command to the magnetic head of the disc for the exact track location of the bucket. Let

t_3 = time needed for locating the bucket to seek time of the disc pack.

In the MFS_2 discussed in section 4, each bucket contains $s(s-1)/2$ sub-buckets. The sub-buckets will correspond to small segments on a track with sub-bucket labels. Thus, the computer can give a direct command to the magnetic head to search a particular sub-bucket and retrieve all the accession numbers in it. Let

t_4 = time needed to locate a sub-bucket and retrieve the accession numbers of pertinent query search time on a track.

At times, sub-buckets may be chained. In such situations, the chaining links can be picked up by the magnetic head, ordered, and the retrieval procedure completed in another rotation of the disc. Let

t_5 = time needed for tracking chaining, if required.

Thus, $$T_1 = t_1+t_2+t_3+t_4+t_5 \qquad (6.1)$$

T_2 will depend upon the number of records pertinent to a query which will depend on the distribution of the records in the file. If the distribution of the records is uniform and the distribution of usage of queries is uniform, then an expression for the upper bound of average T_2 may be obtained as follows:

Let

[illegible] = the average seek time on a disc.

[illegible] = the average search time on a disc.

$\bar{S}_{j_1 j_2}$ = {set of all the ℓ attributes except A_{j_1} and A_{j_2}}.

The average number of records satisfying a query

$$\bar{n}_q = \frac{2}{\ell(\ell-1)} \sum_{j_1}^{\ell} \underset{\neq}{} \sum_{j_2}^{\ell} \prod_{i \in \bar{S}_{j_1 j_2}} n_i \qquad (6.2)$$

where the double summation is over all $\lambda(\lambda-1)/2$ values of j_1 and j_2 from 1 to λ, where $j_1 \neq j_2$.

As more than one pertinent record may be on the same track, hence

$$\text{Average } T_2 \le (\tau+\omega)\, \tilde{n}_q. \tag{6.3}$$

Thus, the average retrieval time for a query under all the stated assumptions is

$$T_1 + \text{Average } T_2.$$

7. MULTIPLE VALUED FILING SCHEMES AND PARTIALLY DELETED GEOMETRY

Consider an EG(N, s) and the spread $a_1x_1+a_2x_2+\ldots+a_Nx_N = c$ where $a_i \in GF(s)$ and are fixed and c varies over all elements of GF(s). Let $n_1, n_2, \ldots, n_\ell$ be a set of positive integers with $\ell \le s$, and $\max\{n_i\} \le s^{N-1}$.

Consider the element $a_1x_1 + a_2x_2 + \ldots + a_Nx_N = \alpha_i$ ($i=0,1,2, \ldots s-1$, $\alpha_i \in GF(s)$)of the spread and delete from it $s^{N-1}-n_{i+1}$ points. Then delete all lines which lie completely on any one of the elements of the spread. Some lines which do not lie completely on an element of the spread may be deleted because some of the points have been deleted, and at least two points are needed to define a line. There are some lines in this geometry which may have less than s points. This partially deleted geometry has $\sum_{i=1}^{s} n_i$ points. The number of lines in this geometry depends on the n_i's and also on the actual points which are retained, i.e., for a fixed set of n_i's the number of lines may vary depending on the choice of the points which are not deleted. Such properties of partially deleted geometries are not very well known and no attempt will be made in this paper to develop such theories; but some interesting results which have been obtained by simulating on the computer will be stated later.

Consider a file with s attributes and the ith attribute has n_i, ($n_i > 0$, $i = 1,2,\ldots,s$) distinct values. Associate the ith attribute, A_i, with the element $a_1x_1+a_2x_2\ldots+a_Nx_N = a_{i-1}$ of the spread in the partially deleted geometry. The n_i points on this element are associated with the n_i values of the attribute A_i. The lines of this partially deleted geometry are associated with the buckets of the filing scheme. The storing rule for the accession numbers of the records in the buckets is the same as before. The sub-bucket may be constructed in a similar manner as before. The exact number of buckets in the filing scheme cannot be stated. It is obvious that $b \le s^{2(N-1)}$ and it is difficult to state a lower bound for b. Given a set of n_i's, there are some practical situations in which the minimum number of buckets are preferred, whereas in some other situations, maximum number of buckets are preferred. Some simulations were performed on the computer and some interesting numerical results were obtained which are given in the following example.

Example 7.1

Consider an EG(3,5). This geometry has 125 points, 775 lines and 155 planes. The planes can be divided into 31 groups of 5 each, where each group represents a spread of the geometry. A partially deleted geometry is constructed by deleting the following 38 points: 000, 002, 004, 010, 012, 014, 021, 022, 023, 030, 034, 100, 102, 104, 110, 111, 113, 120, 122, 124, 142, 231, 223, 300, 304, 323, 340, 344, 412, 421, 423, 424, 430, 432, 434, 441, 442, 443. This partially deleted geometry has 150 lines with 2 points each, 253 lines with 3 points each, 222 lines with 4 points each and 150 lines with 5 points each, i.e., deletion of the 38 points has not deleted any line completely. This has been possible because of the particular manner in which the 38 points were chosen. Consider the spread $x_1 = 0$, $x_1 = 1$, $x_1 = 2$, $x_1 = 3$ and $x_1 = 4$. These planes now have 15, 15, 23, 20 and 15 points on them. These 5 planes are now identified with a file with 5 attributes, 3 of which have 15 values each, 1 has 20 values and the other has 23 values. There are 150 lines which lie on these 5 planes. They are deleted from the partially deleted geometry and the remaining

625 lines are identified with the buckets of a filing scheme. Among these 625 lines, there are 110 lines with 2 points on each, 217 lines with 3 points on each, 186 lines with 4 points on each and 112 lines with 5 points on each. Thus, in the filing scheme, there are 110 buckets with 1 sub-bucket in each, 217 buckets with 3 sub-buckets in each, 186 buckets with 6 sub-buckets in each and 112 buckets with 10 sub-buckets in each.

This partially deleted geometry can be used to construct filing schemes for another 20 different types of files by associating them with different spreads. In these files, the different number of values that the attributes can take, i.e., $(n_1, n_2, n_3, n_4, n_5)$, are: (14, 15, 16, 18, 25), (14, 15, 17, 18, 24), (14, 16, 18, 19, 21), (14, 17, 18, 18, 21), (15, 15, 17, 19, 22), (15, 16, 16, 17, 24), (15, 16, 18, 19, 20), (15, 16, 17, 18, 22), (15, 17, 17, 18, 21), (15, 17, 18, 19, 19), (15, 18, 18, 18, 19), (16, 16, 16, 18, 22), (16, 16, 17, 18, 21), (16, 16, 17, 19, 20), (16, 16, 18, 18, 20), (16, 16, 18, 19, 19), (16, 17, 17, 18, 20), (16, 17, 17, 19, 19), (16, 17, 18, 18, 19) and (17, 17, 18, 18, 18).

Thus, the following theorem is established.

<u>Theorem 7.1</u>

There exists a $MVFS_2$ with parameters $t = 2$, $k = s$, $n_1, n_2, \ldots, n_k$, where $n_i \geq 0$ for $i = 1, 2, \ldots, k$ and $b = s^{2(N-1)}$.

Consider a file which has the following type of structure:

$$\begin{array}{ll} \ell_1 \text{ attributes with values between } s^{N-1} \text{ and } s^{N-2}+1 & \\ \ell_2 \text{ attributes with values between } s^{N-2} \text{ and } s^{N-3}+1 & (7.1) \\ \ell_3 \text{ attributes with values between } s^{N-3} \text{ and } s^{N-4}+1 & \\ \cdot & \\ \cdot & \\ \cdot & \\ \ell_{N-1} \text{ attributes with values between } s \text{ and } 0 & \end{array}$$

where $\ell_1, \ell_2, \ldots, \ell_{N-1}$ satisfy the following conditions:

(i) ℓ_i's are non-negative integers:

(ii) $\ell_1 + \left[\frac{\ell_2}{s}\right] + \left[\frac{\ell_3}{s^2}\right] + \ldots + \left[\frac{\ell_{N-1}}{s^{N-2}}\right] \le s.$

For simplicity, it will be assumed that $n_1, n_2, \ldots, n_{\ell_1}$ en s^{N-1} and $s^{N-2}+1$, $n_{\ell_1+1}, n_{\ell_1+2}, \ldots, n_{\ell_1+\ell_2}$ lie between s^{N-2} and $s^{N-3}+1$, $\ldots$, $n_{\ell - \sum_{i=1}^{N-2} \ell_i + 1}$, $\ldots$, n_ℓ lie between s and 0, where ℓ is the total number of attributes.

Consider a EG(N, s) and a (N-1) dimensional spread in it. Choose ℓ_1 elements of this spread and let them be represented by

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1N}x_N = \alpha_0$$

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1N}x_N = \alpha_1 \qquad (7.2)$$

$$\begin{array}{ccc} \vdots & & \vdots \\ a_{11}x_1 + a_{12}x_2 + \cdots + a_{1N}x_N = c_{\ell_1} \end{array} \qquad (7.2)$$

Consider the (N-1) dimensional flat $a_{11}x_1+a_{12}+x_2+ \ldots$ $\ldots +a_{1N}x_N = a_0$ and delete from it $s^{N-1}-n_1$ points and associate the remaining points with the n_1 values of A_1. Similarly, delete $s^{N-1}-n_2$ points from $a_{11}x_1+a_{12}x_2+ \ldots +a_{1N}x_N = a_1$ and associate the remaining points with A_2 and so on up to A_{ℓ_1}.

Let $[{}^{s}i/_{s}i-1] = s_i$. Choose another set of s_2 elements of the spread and partition each element into (N-2) dimensional spreads. These $s_2s \geq \ell_2$ elements may be represented as:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1N}x_N &= a_{\ell_1+j} \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2N}x_N &= c_2 \end{aligned} \qquad (7.3)$$

where $a_{i.}$'s are fixed and c_2 varies over all elements of GF(s). Choose ℓ_2 elements out of (7.3) and delete all the points which lie on the remaining $s_2s - \ell_2$ (N-2) dimensional flats of (7.3). Take one of these remaining elements of (7.3) and delete from it $s^{N-2} - n_{\ell_1+1}$ points and associate the remaining n_{ℓ_1+1} points with the n_{ℓ_1+1} values of A_{ℓ_1+1}. Similarly, delete $s^{N-2} - n_{\ell_1+2}$ points from another element of (7.3) and associate the remaining n_{ℓ_1+2} points with the n_{ℓ_1+2} values of A_{ℓ_1+2} and so on for the attributes $A_{\ell_1+3}, \ldots, A_{\ell_1+\ell_2}$. This process is continued until all the ℓ attributes have been associated with different dimensional flats of the geometry. In this process of association, a partially deleted geometry with $n = \sum_{i=1}^{\ell} n_i$ points is obtained. From this partially deleted geometry all lines which lie completely on any flat, which is associated with an attribute, is deleted. Let us denote the partially deleted geometry by $PDG(N, s, s^N-n)$. The exact number of lines in $PDG(N, s, s^N-n)$ will depend on the s^N-n points which have been deleted. As this deletion can be done in many ways, the number of lines will vary, but will be less than the expression given in equation (4.5).

If the lines of this $PDG(N, s, s^N-n)$ are associated with the buckets of a filing scheme and if the storing rule for the accession numbers of the records are the same as before, then the filing scheme will correspond to a $MVFS_2$ for a file whose structure is given by (7.1). Thus, the following theorem is established.

Theorem 7.2

There exists a $MVFS_2$ with parameters $t = 2$, $\ell = \ell_1+\ell_2+\ldots$ $\ldots+\ell_{N-1}$ where $\ell_i \geq 0$ and ℓ_i attributes have values between s^{N-i} and $s^{N-i-1}+1$, $n_1, n_2, \ldots, n_\ell$ where $n_i \geq 0$ for $i = 1, 2, \ldots, \ell$ and $b \leq s^{N-2}$ $\{s\,\phi(N-1,0,s) - s_1\,\phi(N-2,0,s) - \ldots$ $\ldots - s_{N-2}\,\phi(1,0,s) - s_{N-1}\}$ where s_i $[\ell_i/s^{i-1}]$ and s is a power of a prime integer.

In most practical situations the stucture of a file is not stated as in (7.1) but usually in the following form:

$$\begin{array}{l} \ell_1 \text{ attributes with } v_1 \text{ values} \\ \ell_2 \text{ attributes with } v_2 \text{ values} \\ \vdots \\ \ell_m \text{ attributes with } v_m \text{ values} \end{array} \tag{7.4}$$

where ℓ_i's and v_i's are non-negative integers. The problem is then to find a pair of s and N, and apply theorem 7.2. For simplicity, it is assumed that $v_1 > v_2 > \ldots > v_m > 2$. Then s and N are chosen with the following properties:

(i) s is a power of a prime integer.

(ii) There exist m pairs of integers (ℓ_1, N), (s_2, N_2), (s_3, N_3), ..., (s_m, N_m) which satisfy the following conditions:

$\ell_1 < s$, s^N is the smallest integer which exceeds v_1

$s_2 s^{N_2} > \ell_2$ and s^{N-N_2} is the smallest integer which exceeds v_2

$s_3 s^{N_3} > \ell_3$ and s^{N-N_3} is the smallest integer which exceeds v_3

$\vdots$

$s_m s^{N_m} > \ell_m$ and s^{N-N_m} is the smallest integer which exceeds v_m

and $\ell_1 + s_2 + s_3 + \ldots + s_m = s$

and $N_i < N$ for $i = 2, 3, \ldots, m$.

Having chosen N and s as satisfying the above conditions, it is easy to see that theorem 7.2 can be applied to construct the $MVFS_2$ for the file structure given in (7.4).

8. ACKNOWLEDGEMENT

The author would like to thank Drs. C. P. Wang and F. P. Palermo of the IBM Research Laboratory for valuable discussions during this work.

9. REFERENCES

1. Abraham, C. T., Ghosh, S. P. and Ray-Chaudhuri, D. K., (1968) "Filing Schemes Based on Finite Geometries," Information and Control Vol. 12, No. 2, pp. 143-163.

2. Abraham, C. T. Bose, R. C. and Ghosh, S. P. (1967), "File Organization of Records for Multiple Valued Attributes for Multivariate Queries," IBM Research Report No. 1886.

3. Abraham, C. T., Bose, R. C. and Ghosh, S. P. (1967), "File Organization of Records with Unequal Valued Attributes for Multi-attribute Queries," preliminary report to Air Force Contract No. AF 30(602)-4088, May, 1967.

4. Bose, R. C. (1947), "Mathematical Theory of Symmetrical Factorial Designs," Sankhya, Vol 8, pp. 107-166.

5. Buchholz, W. (1963), "File Organization and Addressing," IBM Systems Journal Vol. 2, pp. 86-111.

6. Carmichael, R. D. (1937), "Introduction to the Theory of Groups of Finite Order," Ginn and Co., Boston, Mass.

7. Davis, D. R. and Lin, A. D. (1965), "Secondary Key Retrieval Using an IBM 7090-1301 System," Communications of the ACM Vol 8, pp. 243-246.

8. Demuth, H. B. (1956), "A Report on Electronic Data Sorting," Stanford Research Institute.

9. Ghosh, S. P. and Abraham, C. T. (1968), "Application of Finite Geometry in File Organization for Records with Multiple-valued Attributes," IBM Journal of Research and Development Vol. 12, No. 2, pp. 180-187.

10. Ghosh, S. P. (1968), "On the Problem of Query-oriented Filing Schemes Using Discrete Mathematics," Proceedings of the IFIP Congress '68, Edinburgh, Scotland, pp. F74-79.

11. Ray-Chaudhuri, D. K. (1968), "Combinatorial Information Retrieval Systems for Files," SIAM Journal on Applied Math Vol. 16, No. 5, pp. 973-992.

SECTION V

CALIBRATION OF THE FILE ORGANIZATION EVALUATION MODEL (FOREM I)

H. Ling
V. Y. Lum
M. E. Senko

CALIBRATION C THE FILE ORGANIZATION EVALUATION MODEL (FOREM I) *

by

H. Ling
V. Y. Lum
M. E. Senko

Information Sciences Department
IBM Research Laboratory
San Jose, California

ABSTRACT: This section presents the results of the model runs versus actual computer runs to illustrate the kind of accuracy attainable with the model's equation evaluation approach. The average error for the most complex access method tested (ISAM) under the conditions selected is well under ten percent. Generally speaking, the difference between the actual file layout and that of the model accounts for many of the errors.

For the file designer who does not have the resources to run the full FOREM I model, but who wishes to make a quantitative evaluation of a small number of primary key access method designs, we have provided printouts of the relevant FORTRAN subroutines.

FOREM I (FSSM) CALIBRATION

One of the major considerations with any model is its accuracy. This is a particularly crucial consideration in the case of FOREM I which uses equations with complex assumptions to evaluate the elapsed time for the primary key access methods.

In calibrating and testing FOREM I, we have selected the most complex access method available - the IBM Indexed Sequential Access Method (ISAM) - to determine whether accurate equations can be written. The measured results were obtained by John Barlow of SDD using a 360/Mod 50. Different processors will affect the results in some cases. As the tables included in this chapter indicate, the equation evaluation method can be quite accurate; the average error for all experiments is 8.3%. In some experiments, the actual file layout for the overflow area deviated from random; in these cases (which bear asterisks) the runs are shorter than they should be. Removing these cases results in an average deviation of 6.7%.

The results are extremely important for the area of file design because they prove that accurate equations can indeed be written for complex access methods. This means that another avenue of estimating gross performance is open to the file designer. Even if he does not have a computerized model available, he can still hand-calculate timings for typical queries by inserting his parameters into the model equations. In order to make such calculations possible, we have included in this section printouts of the FOREM I FORTRAN access method subroutines.

TEST ENVIRONMENT

CPU - 360/Mod 50

I/O Device - 2314

Access Method - ISAM, master index in core, other indexes and overflow records on same volume as prime records.

File Size - 100,000 records

Record Size - 200 bytes, including 8 bytes key.

Blocking - Full track in prime area, unblocked in overflow area. (30 records/prime track, 20 records/overflow track.)

Records Processed - 5000

CPU Processing Time - 0

NOTATIONS USED IN TABLES

Loading	creation of file.
SR	sequential retrieval.
SSR	skip sequential retrieval.
RR	random retrieval.
RU	random update.
RI	random insertion.
SSI	skip sequential insertion.
cyl	cylinder overflow (overflow records in same cylinder as prime records).
ind	independent overflow (overflow records in different cylinders as prime records).

Mode of Retrieval	Overflow Handling	Percent* Overflow	Model Result (secs.)	Measured Result (secs.)	Model Error (percent)
Loading	ind	0	186.	159.	17.0
SR	cyl	0	10.9***	8.51	28.1
SR	cyl	5.	16.6	16.1	3.11
SR	cyl	16.7	30.0	27.9	7.52
SR	ind	0	10.9***	8.64	26.1
SR	ind	5.	45.5**	36.9	23.3
SR	ind	16.7	82.1**	69.2	18.6
SSR	cyl	0	422.	414	1.93
SSR	cyl	5.	419.	414.	1.21
SSR	cyl	16.7	448.	451.	9.67
SSR	ind	0	422.	412.	2.43
SSR	ind	5.	457.	464.	1.51
SSR	ind	16.7	613.**	544.	12.7
RR	cyl	0	790.	732.	7.92
RR	cyl	5.	787.	744.	5.77
RR	cyl	16.7	816.	773.	5.56
RR	ind	0	781.	715.	9.2
RR	ind	5.	802.	752.	6.64
RR	ind	16.7	922.	846.	8.98
RU	cyl	0	915.	970.	6.01

(cont.)

Mode of Retrieval	Overflow Handling	Percent* Overflow	Model Result (secs.)	Measured Result (secs.)	Model Error (percent)
RU	cyl	5.	912.	951.	4.10
RU	cyl	16.7	941.	999.	5.80
RU	ind	0	906.	955.	5.13
RU	ind	5.	927.	941.	1.49
RU	ind	16.7	1047.	1038.	.87
RI	cyl	0	1422.	1351.	5.25
RI	cyl	5.	1418.	1381.	2.67
RI	cyl	16.7	1446.	1371.	8.16
RI	ind	0	2458.	1937.	26.9
RI	ind	5.	2493.	2005.	24.3
RI	ind	16.7	2717.	2243.	21.1
SSI	cyl	0	1054.	1077.	2.13
SSI	cyl	5.	1050.	1018.	3.14
SSI	cyl	16.7	1078.	963.	11.9
SSI	ind	0	1979.	1957.	1.12
SSI	ind	5.	2027.	1984.	2.17
SSI	ind	16.7	2288.	2207.	3.67

*In order to have the model set-up and the real data set-up be as close as possible, the 0 percent overflow actually has a very small number of overflow records.

(cont.)

**The discrepancy between model and measured results is mainly due to the set-up of overflow records in the created data set. The overflow records belonging to the same track, for example, are stored very close to each other in the actual set-up. In the model, each pair of records is considered to be separated by half as many cylinders as there are overflow cylinders.

***The error in this case is due to the assumption that missing of revolutions occurs when control is returned to CPU to set up the reading of the track index and next data track. In the particular data set chosen, no missing occurs probably because there is a considerable amount of empty space at the end of each data track.

```
C        ***********************************************
C        DEVICE TABLE
C        DV,30,20
C                              *INPUT
C        NO          I         DEVICE NO.
C        BPT                   BYTES/TRACK
C        OBBL                  DEV. OVERHEAD BYTES/BLOCK
C        TAT                   TRACK ACCESS TIME (MS /COMPLETE REV.)
C        TPC                   TRACKS/CYLINDER
C        CAT                   CYL. ACCESS TIME (MS)
C        CPB                   CYL./BIN
C        BAT                   BIN ACCESS TIME
C        TART                  TURNAROUND TIME (MS) TIME TO CHANGE READ/WRITE STATE
C        TDT                   TRACK DEAD TIME (MS)
C                              *COMPUTED AT INPUT TIME
C        PTI         I         DEVICE NUMBER-TO-PTR   CONVERSION
C         KGAP               DEVICE KEY GAP PER BLOCK
C         OFAC               DEVICE OVERHEAD FACTOR PER BLOCK
C
C        ***********************************************
C        FILE TABLE
C        FL,20,50
C        ***********************************************
C                              *INPUT
C        NO          I         FILE NUMBER
C        DEV         I         NUMBER OF DATA AREA DEVICE
C        IDVN        I         NUMBER OF ISAM INDEX DEVICE
C        BIE                   BYTES/INDEX ENTRY
C        AM                    ACCESS METHOD (S=SEQ,D=DIRECT,IE=IS EMBED,IS=SEPAR)
C        TYP                   SI=SEC IND   D= DATA   B= BUCKET
C        RPB                   RECORDS/BLOCK PRIME
C        POV                   PERCENT OF TOTAL FILE IN OVERFLOW AREA
C        OBO                   OVERHEAD BYTES/REC. OVERFLOW AREA
C        PPA                   PER. OF PRIME AREA FILLED
C        POF                   PERCENT OF OVERFLOW AREA FILLED
C        OBB                   FILE OVERHEAD BYTES/BLOCK PRIME
C        MI                    =M IF MASTER INDEX KEPT
C                              *COMPUTED
C        RPPO                  REC/PRIME TRACK IN OVERFLOW AREA
C        RPPT                  RECS/PRIME TRACK INCL PRIME AND OVERFLOW
C        RPTP                  REC/TRACK PRIME
C        BLPT                  BLOCKS/TRACK PRIME
C        RPTO                  REC/TRACK OVERFLOW
C        BLT                   UNROUNDED BLOCKS/TRACK
C        NTP                   TRACKS IN PRIME AREA
C        NCP                   CYLINDERS IN PRIME AREA
C        NBP                   BINS IN PRIME AREA
C        INTI                  TRACKS IN CYLINDER INDEX
C        INCI                  CYLINDERS IN CYLINDER INDEX
C        TNR                   TOTAL RECORDS
C        RSIZ                  RECORD SIZE FOR FILE
C        NTO                   TRACKS OVERFLOW AREA
```

```
C     NCO                CYLINDERS OVERFLOW AREA
C     NBO                BINS OVERFLOW AREA
C     NB                 TOTAL BINS
C     NC                 TOTAL CYLINDERS
C     TNTP               TOTAL TRACKS PRIME
C     IDEV        I      PTR. TO DATA DEVICE
C     IDVI        I      PTR. TO INDEX DEV.
C     ETI         I      FILE NO. TO PTR. CONVERSION
C     SEG         I      PTR TO MASTER SEG OF FILE
C     IFD         I      PTR TO FIELD INDEXED BY FILE
C
C     ***************************************************
C     SEGMENT TABLE
C     SG,30,20
C     ***************************************************
C                        *INPUT
C     NO          I      SEGMENT NUMBER
C     LVL         I      LEVEL
C     SSN         I      SUPERIOR SEGMENT NO.
C     NSS                NO. SEGMENTS PER SUPERIOR SEGMENT
C     FLS         I      NO. OF FILE IT'S ON
C                        *COMPUTED AT INPUT TIME
C     DSN         I      PTR   TO SUPERIOR SEGMENT
C     IMAS        I      PTR   TO MASTER SEGMENT
C     ETI         I      SEGMENT NUMBER-TO-PTR   CONVERSION
C     TNS                TOTAL NO. OF SEGMENTS
C     IFIL        I      PTR   TO FILE IT'S ON
C                        *COMPUTED BY FIELD INPUT
C     SSZ                SEGMENT SIZE
C     RSZ                RECORD SIZE  -  MASTER SEGMENT ONLY
C                        *COMPUTED BY QUERY GENERATOR
C     PSQ                FRACTION OF INDIVIDUAL SEGMENT SATISFYING QUERY
C                        *COMPUTED BY VOLUME (QUALIFIER) ROUTINE
C     PS                 FRACTION QUALIFIED WITH SUBORDINATE QUALIFIERS
C     PG                 FRACTION OF SUPERIOR SEGMENT QUALIFIED BY THIS ONE
C     FCQ                FRACTION COMPLETELY QUALIFIED
C     NCQ                NUMBER COMPLETELY QUALIFIED
C
C     ***************************************************
C     FIELD TABLE
C     FD,100,20
C     ***************************************************
C                        *INPUT   -REAL FIELD                          -PSEUDO FIELD
C     NO          I      FIELD NUMBER                                       FIELD NO
C     SEG         I      NO. OF SEGMENT IT BELONGS TO                          (OMIT
C     CDR         I      PTR   TO DISTRIBUTION OF FIELD VALUES                    O
C     CDQ         I                                 QUERY INITIAL VALS.           M
C     UVR         I                                 UNIQUE QUERY VOLUMES          I
C     QR          I                                 QUERY RANGES                  T
C     TYPE               ORGANIZATION OF FIELD -  S,IS,SI,O                      SI
C     INDN        I      FILE NO. OF  SEC. INDEX IF SI
C     SIZ                FIELD SIZE
```

```
C       UDD        I     PTR. TO DISTRIBUTION OF UPDATE INITIAL VALUES
C       UVD        I                             UNIQUE UPDATE VOLUMES
C       UR         I                             UPDATE RANGES
C                        *COMPUTED AT INPUT TIME
C       ISEG       I     PTR   TO SEGMENT IT BELONGS TO
C       FTT        I     FIELD NUMBER-TO-PTR   CONVERSION
C       IMDI       I     PTR   TO FILE OF SEC INDEX
C                        *COMPUTED BY QUERY GENERATOR
C       PVQ              PERCENT VOLUME QUALIFIED
C       PMQ              PERCENT MASTERS QUALIFIED BY THIS FIELD
C       QDF              QUERY DISTANCE DOWN FILE
C       QRR              QUERY RANGE
C       NMQ              NO. MASTERS QUALIFIED BY FIELD
C
C       **********************************************
C       DISTRIBUTION TABLE
C       DS,60,5,1000
C       **********************************************
C                        *INPUT
C       TYP              =S IF STEP FUNCTION,=I IF INTERPOLATION
C       SIZ        I     NO. OF ENTRIES
C                        *COMPUTED AT INPUT TIME
C       FNT        I     PTR   TO FIRST VALUE IN TABLE
C                        *INPUT
C       VAL              LARGE TABLE FOR VALUES
C
C       **********************************************
C       QUERY/UPDATE TABLE
C        QU,30,10
C       **********************************************
C                        *INPUT
C       FFD        I     FIELD NUMBER
C       MOD              FOR QUERY - Q = QUERY  O = OUTPUT  B = BOTH
C                        FOR UPDATE- Q = QUERY  M = MOD. SI B = BOTH
C                                  A =  MODIFY ALL SI'S TO FILE
C       QUR              U=UNIQUE VALUE QUERY R=RANGE QUERY G= IMMEDIATE QUERY
C       IML              LOW VALUE IMMEDIATE
C       IMH              HIGH VALUE IMMEDIATE
C                        *COMPUTED BY QUERY GENERATOR
C       QIV              INITIAL VALUE
C       QR               RANGE (FOR RANGE QUERIES)
C       DDF              DISTANCE DOWN FILE
C       POV              VOLUME
C
C
C       **********************************************
```

```
C       **************************************************
C       COMPUTE FILE PHYSICAL LAYOUT
        REAL IS,IE
        DATA S,D,IS,IE/2H S,2H D,2HIS,2HIE/
        REAL    IB
        DATA     IB/2HIB/
        DO 60 I=1,NFL
        IF(FLTYP(I).EQ.D)  GO TO  66
        IF ( FLTYP (I) . EQ .SI )  GO TO 77
        READ ( II, 1000) DUP
        WRITE (OT, 1001) DUP
 1000   FORMAT (F10.4)
 1001   FORMAT (/,5X,17HDUP FACTOR               ,5X,F10.4,/ )
        IFD = FLIFD (I)
        ISEG=FDISEG (IFD )
        IMAS = SGIMAS (ISEG )
        FLRSIZ(I)=FDSIZ(IFD)+FLISZ(I)
        FLTNR (I) = DUP * SGTNS (IMAS)
        GO TO 65
   77   IFD  =  FLIFD(I)
        FLRSIZ(I)=FDSIZ(IFD)+FLISZ(I)
        ISEG = FDISEG(IFD)
        FLTNR(I) = SGTNS(ISEG)
        GO TO 65
   66   ISG = FLSEG(I)
        FLRSIZ(I) = SGRSZ(ISG)
        FLTNR(I) = SGTNS(ISG)
   65   IDEV = FLIDEV(I)
        IDVI = FLIDVI(I)
         IF (FLOBB(I).EQ.0.) GO TO 301
         BKSZ =DVOBBL(IDEV) +DVKGAP(IDEV)+CEIL(DVOFAC(IDEV)*(FLRSIZ(I)
     1           *FLRPB(I)+FLOBB(I)))
         BKLAST= DVKGAP(IDEV)+FLOBB(I)+FLRSIZ(I)*FLRPB(I)
         GO TO 302
  301    BKSZ=DVOBBL(IDEV)+CEIL(DVOFAC(IDEV)*FLRSIZ(I)*FLRPB(I))
         BKLAST=FLRSIZ(I)*FLRPB(I)
  302    IF(BKLAST.GT.DVBPT(IDEV)) GO TO 303
         BLN=FLOOR((DVBPT(IDEV)-BKLAST)/BKSZ)
         FLBLPT(I)=BLN+1.
         REM=DVBPT(IDEV)-BLN*BKSZ-BKLAST
         FLBLT(I)=FLBLPT(I)+REM/BKSZ
         FLRPTP(I)=FLOOR(FLBLPT(I)*FLRPB(I)*FLPPA(I))
         GO TO 304
  303    FLBLT(I)=(DVBPT(IDEV)+DVOBBL(IDEV))/(BKLAST+DVOBBL(IDEV))
         FLBLPT(I)=1./CEIL(1./FLBLT(I))
         FLRPTP(I)=FLBLPT(I)
  304    IF(FLAM(I).EQ.S) GO TO 61
         IF(FLAM(I).EQ.D) GO TO 61
         IF(FLOBO(I).EQ.0.) GO TO 305
         BKSZO=DVOBBL(IDEV)+DVKGAP(IDEV)+CEIL(DVOFAC(IDEV)*(FLRSIZ(I)
     1         +FLOBO(I)))
         BKLASO=DVKGAP(IDEV)+FLOBO(I)+FLRSIZ(I)
```

```
      GO TO 306
305   BKSZ =DVOBBL(IDEV)+CEIL(DVOFAC(IDEV)*FLRSIZ(I))
      BKLASO= FLRSIZ(I)
306   IF(BKLASO.GT.DVBPT(IDEV)) FLRPTO(I)=1./CEIL((BKLASO+DVOBBL(IDEV))
     1             /(DVBPT(IDEV)+DVOBBL(IDEV)))
      IF(BKLASO.LE.DVBPT(IDEV)) FLRPTO(I)=FLOOR((DVBPT(IDEV)-BKLASO)
     1             /BKSZO) +1.
     IF(FLAM(I).NE.IF) GO TO 63
     IF(FLPOF(I).NE.0.) GO TO 351
      FLNTP(I)=DVTPC(IDEV)-1.
     GO TO 352
351  FLNTP(I) = FLOOR((DVTPC(IDEV)-1.0)/ ((1.0 +(FLPOV(I)*FLRPTP(I))/
     1             ((1.0 - FLPOV(I))*FLRPTO(I)*FLPOF(I)))) )
352  FLNTO(I) = DVTPC(IDEV)-1.0-FLNTP(I)
     FLNCP(I) =CEIL((FLTNR(I)/(FLNTP(I)*FLRPTP(I)))*(1. -FLPOV(I)))
     FLNCPB     =    FLNCP(I)
     FLNBO(I)   = 0.
     FLNBP(I)   = CEIL(FLNCP(I)/DVCPB(IDEV))
     FLNCO(I) = 0.
     FLNCOI     = 0.
     GO TO 64
 63  FLNTP(I) = CEIL((1.0-FLPOV(I))*FLTNR(I)/FLRPTP(I))
     FLNTO(I) = CEIL(FLPOV(I)*FLTNR(I)/(FLRPTO(I)*FLPOF(I)))
     IF(FLAM(I).EQ.IB)  GO TO 69
     FLNCP(I) = CEIL(FLNTP(I)/(DVTPC(IDEV)-1.))
     FLNCPB     =  FLNCP(I)
     FLNCOI      =  CEIL(FLNTO(I)/DVTPC(IDEV))
     FLNCO(I) = FLNCOI
     FLNBO(I)   =   CEIL(FLNCO(I) / DVCPB(IDEV))
     FLNBP(I)   =   CEIL(FLNCP(I) / DVCPB(IDEV))
     GO TO 64
 69  FLNCPB = FLOOR(DVCPB(IDEV)/(1. + (FLPOV(I)*FLRPTP(I)/((1.-FLPOV(I)
    1                                  )*FLRPTO(I)*FLPOF(I)))))
     FLNCO(I)   = DVCPB(IDEV) - FLNCPB
     FLNBO(I)   = 0.
     FLNCOI     = CEIL(FLPOV(I)*FLTNR(I)/(FLRPTO(I)*FLPOF(I)*DVTPC(IDEV)
    1                                                                   ))
     FLNCP(I) = CEIL((1. - FLPOV(I))*FLTNR(I)/(FLRPTP(I)*(DVTPC(IDEV)
    1                                                     - 1.   )))
     FLNBP(I) = CEIL(FLNCP(I)/FLNCPB    )
 64  FLRPPI(I) = FLRPTP(I)/(1.0-FLPOV(I))
     FLRPPO(I)=FLRPTP(I)*FLPOV(I)
     IF(FLRPPO(I).LT.1.) FLRPPO(I)=1.
     FLTNTP(I) = CEIL((1.0-FLPOV(I))*FLTNR(I)/FLRPTP(I))
     FLNB(I)= FLNBO(I)+FLNBP(I)
     FLNC(I)=FLNCP(I) +  FLNCOI
      DKSZ =DVOBBL(IDEV) +DVKGAP(IDEV)+CEIL(DVOFAC(IDEV)*FLBIE(I))
      DKLAST= DVKGAP(IDEV)+FLBIE(I)
      DLN=FLOOR((DVBPT(IDEV)-DKLAST)/DKSZ)
     DLPT=DLN+1.
     FLTNTI(I)  =   CEIL(FLNCP(I)           /DLPT          )
     FLTNCI(I)  =   CEIL(FLTNTI(I)/DVTPC(IDVI))
```

```
      GO TO 60
   61 FLINTP(I) = CEIL(FLINR(I)/FLWRTP(I))
      FLNC(I)   = CEIL(FLINTP(I)/DVTPS(IDEV))
      FLNB(I)   = CEIL(FLNC(I)/DVCPR(IDEV))
   60 CONTINUE
      RETURN
  100 STOP
      END
```

```
C        SAMF
C
C
C
C      - SAM - SEQUENTIAL ACCESS METHOD
       FUNCTION SAM(IFL,PFIL,TP,OB,QU)
C      IFIL   =  PTR TO FILE TO E READ
C      PFIL   =  PCT. OF FILE TO BE READ
C      TP     =  PROCESSING TIME PER RECORD
C      OB     = O  IF OSAM    = B IF BSAM
C      QU     = Q IF QUERY    = U IF UPDATE
C      STARTS READING AT FIRST RECORD OF FILE
C      RETURNS TIME
C      ************************************************
C      DEVICE TABLE
       DIMENSION DVTAB(30,20)
       EQUIVALENCE (DVTAB(1,1),DVNO(1))
       DATA IC,CC/2HIC,2HCC/
       DATA MC/2HMC/
       DATA MAXDV,MAXADV/30,20/
C      ************************************************
       COMMON /DV/ NDV,
C                *INPUT
     1           DVNO                   (30),
     1           DVBPT                  (30),
     1           DVOBBL                 (30),
     1           DVTAT                  (30),
     1           DVTPC                  (30),
     1           DVCAT                  (30),
     1           DVCPB                  (30),
     1           DVBAT                  (30),
     1           DVTART                 (30),
     1           DVTOT                  (30),
     1           DVKGAP         (30),
     1           DVOFAC         (30),
     1            DVCATI             (30),
     1            DVCATL             (30),
C                *COMPUTED AT INPUT TIME
     1           DVETI                  (30)
C
       INTEGER
     1           DVNO                   ,
     1           DVETI
C      ************************************************
C      FILE TABLE
       DIMENSION FLTAB(20,50)
       EQUIVALENCE (FLTAB(1,1),FLNO(1))
       DATA MAXFL,MAXAFL/20,50/
C      ************************************************
       COMMON /FL/ NFL,
C                *INPUT
     1           FLNO                   (20),
      1            FLDEV                           (20),
      1            FLIDVN                          (20),
      1            FLBTF                           (20),
      1            FLAM                            (20),
      1            FLTYP                           (20),
      1            FLRPR                           (20),
      1            FLPDV                           (20),
      1            FLOBO                           (20),
      1            FLPPA                           (20),
      1            FLPOF                           (20),
      1            FLOBR                           (20),
      1            FLISZ           (20),
      1            FLMI                            (20)
       COMMON  /FL/
C                  *COMPUTED
      1            FLRPPD                          (20),
      1            FLRPPT                          (20),
      1            FLRPTP                          (20),
      1            FLBLPT                          (20),
      1            FLRPTO                          (20),
      1            FLBLT                           (20),
      1            FLNTP                           (20),
      1            FLNCP                           (20),
      1            FLNBP                           (20),
      1            FLTNTI                          (20),
      1            FLTNCI                          (20),
      1            FLTNR                           (20),
      1            FLRSIZ                          (20),
      1            FLNTO                           (20)
       COMMON /FL/
      1            FLNCO                           (20),
      1            FLNBO                           (20),
      1            FLNB                            (20),
      1            FLNC                            (20),
      1            FLTNTP                          (20),
      1            FLIDEV                          (20),
      1            FLIDVI                          (20),
      1            FLETI                           (20),
      1            FLSEG                           (20),
      1            FLIFD                           (20)
       INTEGER
      1            FLNO                            ,
      1            FLDEV                           ,
      1            FLIDVN                          ,
      1            FLIDEV                          ,
      1            FLETI                           ,
      1            FLIDVI                          ,
      1            FLSEG                           ,
      1            FLIFD                           ,
       DATA O,B/1HO,1HB/,U/1HU/,D/2H D/
       INTEGER DT
       COMMON/PCTRL/PRINT
      DATA NO/2HNO/
      REAL NO
      DATA OT/6/
      IF(PRINT.EQ.NO) GO TO 1111
      WRITE(OT,100) IFL,PFIL,TP,QB,QQ
 100  FORMAT(50X,9HCALL SAM ,I10,F10.3,F10.3,2(9XA1))
 1111 CONTINUE
      IDEV = FLIDEV(IFL)
      CAT = DVCAT(IDEV)
      TAT = DVTAT(IDEV)
      BAT = DVBAT(IDEV)
      IF(DVTPC(IDEV).EQ.1.) GO TO 1
      TD = FLNC(IFL)*CAT + FLNB(IFL)*BAT
      IF(QQ.EQ.0) GO TO 7
    8 TD = TD + FLINR(IFL)*TAT/FLRPB(IFL)
    7 TD = TD +(FLTNIP(IFL)+.5)*TAT
    9 IF(FLBLPT(IFL).GE.1.0)) GO TO 2
      FIM = (CEIL(1./FLBLPT(IFL))-(1./FLBLPT(IFL)))*TAT
      SC = FLTNR(IFL)-1.
      GO TO 3
    2 FIM = (1. - FLBLPT(IFL)/FLBLT(IFL))*TAT
      SC = FLTNTP(IFL)-1.
    3 XM = TP*FLRPB(IFL) + DVTART(IDEV)
      IF (QB.EQ.0) GO TO 4
      TD = TD + CEIL(XM/TAT)*FLTNR(IFL)*TAT/FLRPB(IFL)
    6 IF(((FIM+DVTDT(IDEV)
     1                    )/TAT - AMOD(XM,TAT)).GT.0.) TD = TD-SC*TAT
    5 SAM= PFIL*TD
      SAM = SAM/1000.
      RETURN
C
C     DSAM
    4 IF(XM.LT.TAT/FLBLT(IFL)) GO TO 5
      TD = TD + CEIL(XM/TAT -1./FLBLT(IFL))*FLTNR(IFL)*TAT/FLRPB(IFL)
      IF((TP*FLRPB(IFL)).GT.(TAT+DVTART(IDEV)))
     1    TD= TD-FLTNR(IFL)*TAT/FLRPB(IFL)
      IF((((TP+(.1*TP))*FLRPB(IFL)).GT.(TAT+DVTART(IDEV))).AND.
     1    (((TP-(.1*TP))*FLRPB(IFL)).LT.(TAT+DVTART(IDEV))))
     2    WRITE(OT,20)
  20  FORMAT(9X,52HTHE RECORD BLOCKING AND PROCESSING TIME ARE CRITICAL)
      IF ((XM/TAT-1./FLBLT(IFL)).GT.0.)GO TO 55
      TD =TD-SC*TAT
       GO TO 5
   55 IF(((FIM+DVTDT(IDEV))/TAT-AMOD(XM/TAT-1.0/FLBLT(IFL),1.0)).GT.0.)
     1 TD = TD -  SC*TAT
      GO TO 5
C
C     SAMT  -  TAPE  SEQUENTIAL ACCESS
    1 TPT = FLRPB(IFL)*TP+DVTART(IDEV)
      IF(QB.EQ.0) TPT= TPT - ((DVBPT(IDEV)/FLBLT(IFL)-DVOBBL(IDEV))*
     1            (TAT/DVBPT(IDEV)))
      IF(TPT.GT.DVTDT(IDEV)) GO TO 16
      GT = DVOBBL(IDEV)*TAT/DVBPT(IDEV)
      GO TO 11
   10 IF(TPI.LT.DVIDT(IDEV)+BAT) GO TO 12
      DT =  TPI - DVIDT(IDEV) - BAT
      BM =  DVBPT(IDEV)*BAT/TAT
      FST = (DVOBBL(IDEV)-BM)*TAT/DVBPT(IDEV)
      GT  = DT + 2.*BAT + FST
      GO TO 11
   12 ADT = TPI - DVIDT(IDEV)
      BM = 2.* ADT*DVBPT(IDEV)*(1. - ADT/(2.*BAT))/TAT
      FST = (DVOBBL(IDEV)-BM)*TAT/DVBPT(IDEV)
      GT =2.*ADT+FST
   11 TO = (((DVBPT(IDEV)/FLBLT(IFL)-DVOBBL(IDEV))*
     1     TAT/DVBPT(IDEV))+GT)*FLTNR(IFL)/FLRPB(IFL)
      GO TO 5
      END
```

```
C         BDAM
C
C
C
C       BASIC DIRECT ACCESS METHOD
        REAL FUNCTION BDAM (IFL,XN,TYP, SO, QO, IP, CN)
C       ***********************************************
C       DEVICE TABLE
        DATA IC,CC/2HIC,2HCC/
        DATA MC/2HMC/
        DIMENSION DVTAB(30,20)
        EQUIVALENCE (DVTAB(1,1),DVNO(1))
        DATA MAXDV,MAXADV/30,20/
C       ***********************************************
        COMMON /DV/ NDV,
C                 *INPUT
      1           DVNO                  (30),
      1           DVRPT                 (30),
      1           DVDBBL                (30),
      1           DVTAT                 (30),
      1           DVTPC                 (30),
      1           DVCAT                 (30),
      1           DVCPR                 (30),
      1           DVBAT                 (30),
      1           DVTART                (30),
      1           DVTOT                 (30),
      1           DVKGAP         (30),
      1           DVOFAC         (30),
      1            DVCAT1             (30),
      1            DVCATL             (30),
C                 *COMPUTED AT INPUT TIME
      1           DVETT                 (30)
C
        INTEGER
      1           DVNO                  ,
      1           DVETT
C       ***********************************************
C       FILE TABLE
        DIMENSION FLTAB(20,50)
        EQUIVALENCE (FLTAB(1,1),FLNO(1))
        DATA MAXFL,MAXAFL/20,50/
C       ***********************************************
        COMMON /FL/ NFL,
C                 *INPUT
      1           FLNO                  (20),
      1           FLDEV                 (20),
      1           FLIDVN                (20),
      1           FLBIE                 (20),
      1           FLAM                  (20),
      1           FLTYP                 (20),
      1           FLRPB                 (20),
      1           FLPDV                 (20),
     1            FLOBO                         (20),
     1            FLPPA                         (20),
     1            FLPOF                         (20),
     1            FLOBB                         (20),
     1            FLISZ             (20),
     1            FLMI                          (20)
      COMMON  /FL/
C                 *COMPUTED
     1            FLRPPO                        (20),
     1            FLRPPT                        (20),
     1            FLRPTP                        (20),
     1            FLBLPT                        (20),
     1            FLRPTO                        (20),
     1            FLBLT                         (20),
     1            FLNTP                         (20),
     1            FLNCP                         (20),
     1            FLNBP                         (20),
     1            FLTNTI                        (20),
     1            FLTNCI                        (20),
     1            FLTNR                         (20),
     1            FLRSIZ                        (20),
     1            FLNTO                         (20)
      COMMON /FL/
     1            FLNCO                         (20),
     1            FLNBO                         (20),
     1            FLNB                          (20),
     1            FLNC                          (20),
     1            FLTNTP                        (20),
     1            FLIDEV                        (20),
     1            FLIDVI                        (20),
     1            FLETI                         (20),
     1            FLSEG                         (20),
     1            FLIFD                         (20)
      INTEGER
     1            FLNO                          ,
     1            FLDEV                         ,
     1            FLIDVN                        ,
     1            FLIDEV                        ,
     1            FLETI                         ,
     1            FLIDVI                        ,
     1            FLSEG                         ,
     1            FLIFD
      DATA XI,C/1HI,1HC/
      COMMON/PCTRL/PRINT
      REAL NO
      DATA NO/2HNO/
      REAL LENGTH
      ABQ=0.
      IDEV=FLIDEV(IFL)
      INTEGER OT
      DATA OT/6/
      TATO=DVTAT (IDEV)
      CATD=DVCAT(IDEV)
      BATD=DVBAT(IDEV)
      PPA= FLPPA(IFL)
      IF ( QU.NE.XI) GO TO 1
      PXN = XN / FLINK(IFL)
      IF ( PXN .LT. 0.05) GO TO 1
      I=1
   11 ABQ= LENGTH ( PPA, FLRPB(IFL) ) +ABQ
      IF(PRINT.EQ.NO) GO TO 1112
      WRITE (OT,100 ) PPA ,FLRPB(IFL),ABQ
 1112 CONTINUE
      FI=I
      PIXN = FI * 0.05
      IF ( PIXN. GE. PXN) GO TO 12
      I= I+1
      PPA = PPA+ 0.05
      GO TO 11
  100 FORMAT (1H0 , 7HFLPPA=.,  F10.3,5X,7HFLRPB=    , F10.3 ,7HFLABQ,
     1      F10.3 ,// )
   12 FLABQ = ABQ / FI
      GO TO 2
    1 FLABQ=LENGTH(PPA,FLRPB(IFL))
    2 ICON = FLPOF(IFL)
      ICON=0.
      FLABQ=1.
      IF(PRINT.EQ.NO) GO TO 1114
      WRITE (OT,100 ) PPA ,FLRPB(IFL), FLABQ
 1114 CONTINUE
      TD =XN* ICON +CATD+BATD+ (XN-1.)*(BATD*(1.-1./FLNB (IFL))
     1     + CATD*(1.-1./FLNC (IFL)) )
      IF ( FLBLPT(IFL).LT. 1.) GO TO 3
      IF ( FLMI(IFL).NE.T ) GO TO 3
      WT = 1.
      ST = .5
      GO TO 4
    3 WT = .5
      ST = 0.
    4 TD = TD+XN* (WT+FLABQ/FLBLPT(IFL)) * TATD
      TD = TD+XN* (CEIL(ST+FLABQ/FLBLPT(IFL)) -1.) * (BATD /
     1     (DVTPC(IDEV) * DVCPB(IDEV) )+ CATD/DVTPC(IDEV) )
      IF ( QU.NE.XI) GO TO 5
      TD = TD + XN*2.* CEIL(1./FLBLPT(IFL))* TATD
    5 BDAM = TD+XN* TP
      IF (QU.EQ.C ) BDAM= BDAM +TATD* CN*2.*CEIL(1./FLBLPT(IFL))
      BDAM = BDAM/1000.
      IF(PRINT.EQ.NO) GO TO 1111
      WRITE (OT, 101 ) BDAM
  101 FORMAT ( //, 5X , 5HBDAM= , 5X, F14.4 )
 1111 CONTINUE
      RETURN
      END
C       LENGTH
```

```
C
C
      REAL FUNCTION LENGTH(SIZ,NO)
      REAL NO
      DIMENSION SOTAB(30,8)
      DATA SOTAB /
     1.05,2.,.1,3.,.2,4.,.3,5.,.4,5.,.5,6.,.6,6.,.65,7.,.7,7.,.75,7.,.8,
     A7.,.85,7.,.9,7.,.95,7.,1.0,7.,
     21.,1.,1.,1.053,1.,1.137,1.,1.23,1.,1.366,1.,1.541,1.,1.773,1.,2.0,
     B1.,2.26,1.,2.7,1.,3.223,1.,4.3,1.,5.52,1.,11.,1.,16.91,
     32.,1.,2.,1.,2.,1.05,2.,1.12,2.,1.20,2.,1.24,2.,1.32,2.,1.4,2.,
     C1.50,2.,1.65,2.,1.85,2.,2.5,2.,3.2,2.,5.2,2.,14.91,
     42*0.,5.,1.,5.,1.,5.,1.02,5.,1.04,5.,1.08,5.,1.12,5.,1.14,5.,1.18,
     D5.,1.25,5.,1.35,5.,1.5,5.,1.85,5.,2.6,5.,11.,
     54*0.,10.,1.,10.,1.,10.,1.,10.,1.02,10.,1.04,10.,1.05,10.,1.06,10.,
     E1.08,10.,1.15,10.,1.2,10.,1.35,10.,1.85,10.,8.,
     66*0.,20.,1.,20.,1.,20.,1.,20.,1.,20.,1.01,20.,1.02,20.,1.03,20.,
     F1.05,20.,1.08,20.,1.15,20.,1.50,20.,7.,
     710*0.,50.,1.,50.,1.,50.,1.,50.,1.,50.,1.,50.,1.,50.,1.,50.,1.03,
     G50.,1.15,50.,4.,
     814*0.,100.,1.,100.,1.,100.,1.,100.,1.,100.,1.,100.,1.,100.,1.,
     F100.,1. /
      NTAB= 30
       A=0.
    7 DO 1 I2=1,NTAB,2
      IF(SOTAB(I2,1).GT.SIZ)GO TO 2
    1 CONTINUE
    2 IF(I2.EQ.1)I2 = 3
      NE = SOTAB(I2+1,1)  +1.
      DO 3 J2 = 2,NE
      IF (SOTAB(I2,J2).GT.NO) GO TO 4
    3 CONTINUE
    4 NE=SOTAB(I2-1,1) +1.
      DO 5 K2=2,NE
      IF(SOTAB(I2-2,K2).GT.NO)GO TO 6
    5 CONTINUE
    6 IF(J2.EQ.2) J2 =3
      IF(K2.EQ.2) K2=3
      T1 = ((NO - SOTAB(I2-2,K2-1))/(SOTAB(I2-2,K2)-SOTAB(I2-2,K2-1)))
     1*(SOTAB(I2-1,K2)-SOTAB(I2-1,K2-1))+SOTAB(I2-1,K2-1)
      T2 = ((NO - SOTAB(I2,J2-1))/(SOTAB(I2,J2)-SOTAB(I2,J2-1)))
     1*(SOTAB(I2+1,J2)-SOTAB(I2+1,J2-1))+SOTAB(I2+1,J2-1)
      LENGTH=A + ((SIZ -SOTAB(I2-2,1))/(SOTAB(I2,1) - SOTAB(I2-2,1)))
     1*(T2-T1)+T1
      RETURN
      END
```

```
C        ISAMF
C
C
C
C       ISAM  INDEX SEQUENTIAL ACCESS METHOD
        REAL FUNCTION ISAM(IFL,XN,TYP,SO,QU,TP,CN,BONO)
C       IFL  =  PTR TO FILE
C       XN   =  NO. RECORDS TO BE RETRIEVED
C       TYP  =  Q  IF QISAM  I.E.  XN  CONSECUTIVE RECORDS
C            =  S  IF SISAM  XN RECORDS INDEPENDENTLY BY KEY
C       SO   =  S  IF SISAM KEYS SORTED
C            =  U  IF SISAM KEYS UNSORTED
C       QU   =  I  IF INSERT TYPE UPDATE
C            =  C  IF MODIFY TYPE UPDATE
C            =  Q  IF QUERY
C       TP   =  PROCESSING TIME  PER RECORD (MS)
C       CN   =  NO. RECORDS COMPLETELY QUALIFYING
C
C
C       BUFFERING OPTION - SYSTEM PARAMETER
C       RT RECORD TRACK ONLY
C       TX + TRACK INDEX
C       CI + CYLINDER INDEX
C       MI + MASTER INDEX
C       CEIL(X)  = REAL  'CEILING FUNCTION' - SMALLEST INTEGER GREATER
C                   THAN OR EQUAL TO X
        DATA BO/2HRT/
        DIMENSION FLNCI(20)
        EQUIVALENCE (FLNCI(1),FLTNCI(1))
        DATA IC,CC/2HIC,2HCC/
         REAL MC,IC
        DATA MC/2HMC/
C       *******************************************
C       DEVICE TABLE
        DIMENSION DVTAB(30,20)
        EQUIVALENCE (DVTAB(1,1),DVNO(1))
        DATA MAXDV,MAXADV/30,20/
C       *******************************************
        COMMON /DV/ NDV,
C                  *INPUT
      1            DVNO                 (30),
      1            DVBPT                (30),
      1            DVOBBL               (30),
      1            DVTAT                (30),
      1            DVTPC                (30),
      1            DVCAT                (30),
      1            DVCPB                (30),
      1            DVBAT                (30),
      1            DVTART               (30),
      1            DVTOT                (30),
      1            DVKGAP        (30),
      1            DVOFAC        (30),
     1          DVCATI            (30),
     1          DVCATL            (30),
C               *COMPUTED AT INPUT TIME
     1          DVETI                 (30)
C
      INTEGER
     1          DVNO                  ,
     1          DVETI
C     ****************************************************
C     FILE TABLE
      DIMENSION FLTAB(20,50)
      EQUIVALENCE (FLTAB(1,1),FLNO(1))
      DATA MAXFL,MAXAFL/20,50/
C     ****************************************************
      COMMON /FL/ NFL,
C               *INPUT
     1          FLNO                  (20),
     1          FLDEV                 (20),
     1          FLIDVN                (20),
     1          FLBIE                 (20),
     1          FLAM                  (20),
     1          FLTYP                 (20),
     1          FLRPB                 (20),
     1          FLPOV                 (20),
     1          FLOBO                 (20),
     1          FLPPA                 (20),
     1          FLPOF                 (20),
     1          FLOBF                 (20),
     1          FLISZ         (20),
     1          FLMI                  (20)
      COMMON  /FL/
C               *COMPUTED
     1          FLRPPD                (20),
     1          FLRPPT                (20),
     1          FLRPTP                (20),
     1          FLBLPT                (20),
     1          FLRPTD                (20),
     1          FLBLT                 (20),
     1          FLNTP                 (20),
     1          FLNCP                 (20),
     1          FLNBP                 (20),
     1          FLTNTI                (20),
     1          FLTNCI                (20),
     1          FLTNR                 (20),
     1          FLRSIZ                (20),
     1          FLNTO                 (20)
      COMMON /FL/
     1          FLNCO                 (20),
     1          FLNBO                 (20),
     1          FLNB                  (20),
     1          FLNC                  (20),
     1          FLTNTP                (20),
     1           FLIDEV                  (20),
     1           FLIDVI                  (20),
     1           FLETI                   (20),
     1           FLSEG                   (20),
     1           FLIFD                   (20)
      INTEGER
     1           FLNO                    ,
     1           FLDEV                   ,
     1           FLIDVN                  ,
     1           FLIDEV                  ,
     1           FLETI                   ,
     1           FLIDVI                  ,
     1           FLSEG                   ,
     1           FLIFD
      COMMON/PCTRL/PRINT
      DATA NO/2HNO/
      REAL NO
      DATA RI,IX,CI/2HRI,2HII,2HCI/
      REAL IB,I,IE
      DATA IB, I,IE/2HIB,2H I,2HIE/
      REAL MI,M,IS
      DATA MI,M/2HMI,2H M/
      DATA   C/2H C/
      DATA B/1HB/
      DATA BN/6H      /
      DATA S, O,U,IS,SI,UI/1HS,2H O,1HU,2HIS,2HSI,2H I/
      INTEGER OT
      DATA OT/6/
      IF(TYP.EQ.0) GO TO 1
      IF(PRINT.EQ.NO) GO TO 1111
      WRITE(OT,100) IFL,XN,TYP,SO,OO,IP,CN
  100 FORMAT(50X,9HCALL ISAM,I10,F10.0,9XA1,9XA1,8XA2,F10.3,F10.0//)
 1111 CONTINUE
      TI=0.
      TO=0.
      IDV  = FLIDEV(IFL)
      IDEV = FLIDEV(IFL)
      IDVI  =  FLIDVI(IFL)
      TATO= DVTAT(IDV)
      BATO = DVBAT(IDV)
      CATO= DVCAT(IDV)
      TATI= DVTAT(IDVI)
      CATI= DVCAT(IDVI)
      CATOI=DVCATI(IDV)
      CATOL=DVCATL(IDV)
      CATII=DVCATI(IDVI)
      CATIL=DVCATL(IDVI)
      CATF=CYL((FLNCP(IFL)/2.),IDV)
      CATIN=CYL((FLNCI(IFL)/(2.          )),IDVI)
      CATIF=CYL((FLNCI(IFL)+FLNCP(IFL))/2.,IDV)
      TNR  = FLTNR(IFL)
      TNRCO = FLOOR(QUAL(XN/TNR,FLRPB(IFL),
     1  FLTNR(IFL)*(1.-FLPOV(IFL))*CEIL(FLBLPT(IFL)*FLPPA(IFL))
     1 /FLRPTP(IFL)))
      IF(FLRPPO(IFL).EQ.1.) TNUCO = XN*FLPOV(IFL)
      IF(FLRPPO(IFL).NE.1.) TNUCO = QUAL(XN/TNR,FLRPPO(IFL),FLTNIP(IFL))
      TNTO = QUAL(XN/TNR,FLRPPT(IFL),FLTNIP(IFL))
      IF(FLAM(IFL).EQ.IE) S1 = FLNTP(IFL)
      IF(FLAM(IFL).NE.IE) S1 = DVIPC(IDEV) -1.
      TNCO = QUAL(TNTO/FLTNIP(IFL),S1          ,FLNCP(IFL))
      IF(FLAM(IFL).EQ.IB)
     1  S2= FLOOR(DVCPB(IDEV)/(1.+ (FLPOV(IFL)*FLRPTP(IFL)/(((1.-FLPOV
     1      (IFL))*FLRPTO(IFL)*FLPOF(IFL))))))
      IF(FLAM(IFL).NE.IB) S2 = DVCPB(IDEV)
      TNBO = QUAL(TNCO/FLNCP(IFL),S2           ,FLNBP(IFL))
      TNIOI= QUAL(TNCO/FLNCP(IFL),FLNCP(IFL)/FLTNTI(IFL),FLTNTI(IFL))
      TNCOI= QUAL(TNIOI/FLTNTI(IFL),DVIPC(IDVI),FLTNCI(IFL))
      IF(PRINT.EQ.NO) GO TO 1112
      WRITE (OT,1999)
 1999 FORMAT(30X5HTNBLO,9X,5HTNUCO,9X,4HTNTO,10X,4HTNCO,10X,4HTNBO,
     1  10X,4HTNOI,10X,5HTNCOI)
      WRITE(OT,2000) TNBLO,TNUCO,TNTO,TNCO,TNBO,TNIOI,TNCOI
 2000 FORMAT (26X,7F14.4)
 1112 CONTINUE
      IF(TYP.EQ.B) GO TO 209
C     SINGLE ISAM
      IF(SO.EQ.S) GO TO 2
      IF(BO.NE.CI.AND.BO.NE.MI) GO TO 3
      CITA = XN*(1. - 1./FLTNTI(IFL))
      GO TO  4
    3 CITA = XN*(1. - 1./FLNCP(IFL))
   4  IF(IDEV.EQ.IDVI) GO TO 5
      CICA = XN*(1.-1./FLTNCI(IFL))
      GO TO 6
    5 CICA = CITA
      GO TO 6
    2 IF(BO.NE.CI.AND.BO.NE.MI) GO TO 7
      CITA = TNIOI
      CICA = TNIOI
      GO TO  8
    7 CITA = TNCO
      CICA = TNCO
   8  IF(IDEV.EQ.IDVI) GO TO 6
      CICA=TNCOI
      WRITE(6,7113)CATF,CATIN
 7200 FORMAT(30X,5E15.5)
 7201 FORMAT(20X,5E16.5)
 7113 FORMAT(40X,2E20.5)
 7111 FORMAT ( 10X,F15.5, 10X,F15.5)
    6 WRITE (6,7200) CITA,CICA,CATF,CATIN,CATIF
      IF ((FLMI(IFL).EQ.M).OR.(FLMI(IFL).EQ.MC)) GO TO 9
      IF (FLMI(IFL).EQ.CC) GO TO 11
      TI=CATD+(ONE(CICA)-1.)*CYL((FLNCI(IFL)+FLNCP(IFL))/2.,IDEV)
     1+(FLOOR(FLTNCI(IFL)/2.))*CICA*CATI1+(     (FLTNTI(IFL)/2.))*CITA
   2 *TATO
      IF (IDEV.NE.IOVI)  TI=CATI+(ONE(CICA)-1.)*CYL(FLNCI(IFL)/2.,IOVI)+
     1(FLOOR(FLINCI(IFL)/2.))*CICA*CATII+(     FLTNTI(IFL)/2.)*CITA*TATI
      GO TO 11
    4 WRITE(6,7201)CITA,CICA,CATF,CATIN,CATIF
      IF (FLNI(IFL).EQ.MC) GO TO 205
      IF (BO.EQ.MI) GO TO 12
      TI=CATO+(ONE(CICA)*(1.+(1.-1./FLNCI(IFL)))-1.)*CATIN+2.*CITA*TATO
      IF (IDEV.NE.IOVI)  TI=CATI +(ONE(CICA)*(1.+(1.-1./FLNCI(IFL)))-1.)
     1*CATIN+2.*CITA*TATO
      GO TO 11
   12 TI=CATO+(ONE(CICA)-1.)*CATIF+1.0*CITA*TATO+1.0*TATO
      IF(IDEV.NE.IOVI) TI=CATI+(ONE(CICA)-1.)*CATIN+1.0*(CITA+1.)*TATI
      GO TO 11
  205 WRITE(6,7200)CITA,CICA,CATF,CATIN,CATIF
      IF (IDEV.EQ.IOVI) GO TO 206
      TI=CATI+(ONE(CICA)-1.)*CATIN+1.0*CITA*TATI
      GO TO 11
  206 TI=CATO+(ONE(CICA)-1.)*CATF+1.5*CITA*TATO
      GO TO 11
   11 WRITE (6,7111)  TI, TO
      IF(SO.EQ.S)  GO TO 13
      TO =  BATO*      (1.0+(XM-1.0)*(1.0-1.0/FLNBP(IFL)))
     1        +CATF*      (1.0+(XN-1.0)*(1.0-1.0/FLNCP(IFL)))
      GO TO  14
   13 IF (FLMI(IFL).EQ.CC) GO TO 369
      IF(IDEV.EQ.IOVI) GO TO 131
  369 IF(FLNBP(IFL).LE.1.) GO TO 132
      IF (TNCO      .GT.FLNBP(IFL)) GO TO 133
      TO=    TNBO*BATO+TNCO*CATO
      GO TO 15
  131 TO=    TNBO*BATO+TNCO*CATIF
      GO TO 15
  132 TO=    TNBO*BATO+TNCO*CYL((FLNCP(IFL)/TNCO),IDEV)
      GO TO 15
  133 TO=    TNBO*BATO+FLNBP(IFL)*CATO+(TNCO-FLNBP(IFL))
     1    *CYL((FLNCP(IFL)/TNCO),IDEV)
   15 WRITE (6,7111)  TI, TO
      IF(BO.NE.BN.AND.BO.NE.RI) GO TO 18
      TO = TO + TNTO*1.0*TATO
      GO TO 17
   14 WRITE (6,7111)  TI, TO
      IF(BO.NE.BN.AND.BO.NE.RI) GO TO 16
      TO =  TO+(1.0+(XN-1.0)*(1.0-1.0/FLTNTP(IFL)))*1.0*TATO
      GO TO 17
   16 TO= TO + (1.+(XN -1.)*(1.-1./FLNCP(IFL)))*1.0*TATO
      GO TO 17
   18 TO = TO + TNCO*1.0*TATO
  222 FORMAT(20H0CYLINDER INDEX TIME   ,5X,16HTRACK INDEX TIME    )
  223 FORMAT (2(10X,E20.8))
   17 WRITE (6,7111)  TI, TO
      TEMP1=TO
      IF (FLBLPT(IFL).GT.1.) GO TO 195
      TD=TD+XN*(1.-FLPOV(IFL))*(1./FLBLT (IFL))*TATD
      TEMP2=TD-TEMP1
      GO TO 191
  195 TD=TD+XN*(1.-FLPOV(IFL))*((FLBLPT(IFL)-1.)/(2.*FLBLPT(IFL))
     1+1./FLBLT (IFL)+0.5)*TATD
      TEMP2=TD-TEMP1
      GO TO 191
224   FORMAT (18HOPRIME ACCESS TIME          )
225   FORMAT (5X, E20.8)
  191 WRITE (6,7111)  TI, TD
      WRITE (6,7111) XN,FLRPTO(IFL)
      IF(FLAM(IFL).NE.IE) GO TO 19
      IF(FLRPTO(IFL).GT.1.) GO TO 4019
                                 TD=TD+XN*(FLPOV(IFL)*TATD)*((1./FLRPTO(IFL)
     1)+((FLRPPO(IFL)-1.)/2.)*CEIL((1./FLRPTO(IFL)))+.5)
      GO TO 4017
4019  TD=TD+XN*FLPOV(IFL)*TATD*( ( (FLRPPO(IFL)-1.)/2.)+(0.5+1./FLRPTO
     1(IFL))   )
4017  WRITE (6,7111)  TI, TD
      IF ((QU.NE.UI).AND.(QU.NE.IC)) GO TO 20
      WRITE (6,7111) FLRPTO(IFL), XN
      WRITE (6,7111)  TI, TD
       WRITE (6,7200) XN,FLPOV(IFL),FLRPPO(IFL),FLRPTO(IFL),TATD
      IF(FLPOV(IFL).EQ.0.) GO TO 21
      IF(FLBLPT(IFL).LE.1.) GO TO 4004
      IF (QU.NE.IC) GO TO 4020
      TD=TD+XN*TATD*((1.-FLPOV(IFL))*(5.5
     1                                  +(1./FLRPTO(IFL))+((FLBLPT(IFL)-
     11.)/2.)* (1./FLBLPT(IFL) +3.*CEIL (1./FLBLPT(IFL)))+3.)+7.*FLPOV
     2(IFL))
      GO TO 20
4020                      TD=TD+XN*TATD*((1.-FLPOV(IFL))*(
     1                                                2.5+(1./FLRPTO(IFL))
     1+((1./FLBLT(IFL) )+2.*CEIL(1./FLBLPT(IFL)))*((FLBLPT(IFL)-1.)/2.)+
     2 2.)+FLPOV(IFL)*4.)
      GO TO 20
4004  IF (QU.NE.IC) GO TO 4021
      TD=TD+XN*TATD*((1.-FLPOV(IFL))*(2.*CEIL(1./FLBLPT(IFL))+7.+
     1(1./FLRPTO(IFL))+
     1CEIL(1./FLRPTO(IFL))+(CEIL(1./FLBLPT(IFL))-(1./FLBLPT(IFL))))
     2    +FLPOV(IFL)*(5.*CEIL(1./FLRPTO(IFL))+4.) )
      GO TO 20
4021                       TD=TD+XN*TATD*((1.-FLPOV(IFL))*(CEIL(1./FLBLPT(IFL)
     1)+4.+     (1./FLRPTO(IFL))+(CEIL(1./FLBLPT(IFL))-(1./FLBLPT(IFL))))
     2+FLPOV(IFL)
     2             *(3.*CEIL(1./FLRPTO(IFL))+2.))
      GO TO 20
   19 IF (FLPOV(IFL).EQ.0.) GO TO 1004
      IF(FLAM(IFL).EQ.IB) GO TO 141
      TD=TD+FLPOV(IFL)*(XN*(1.+((FLRPPO(IFL)
     1                                    -1.)/2.)*(1.-(1./FLNBO(IFL))))
```

```
    1    +(XN-1.))*BATD
     IF(SU.EQ.S) GO TO 142
     TD=TD-(XN-1.)*FLPOV(IFL)*(1.-1./FLNBP(IFL))*BATD
     GO TO 143
 142 TD=TD-BATD*FLPOV(IFL)*(INBO-1.)
 143 WRITE (6,7111) II, TD
     X=(FLNCO(IFL)/DVCPB(IDV)-FLOOR(FLNCO(IFL)/DVCPB(IDV)))*DVCPB(IDV)
     IF (FLNCO(IFL).GT.DVCPB(IDV)) GO TO 331
     CATD=CYL(X/2.,IDV)
     GO TO 332
 331 CATD=(DVCPB(IFL)/FLNCO(IFL))*(FLOOR(FLNCO(IFL)/DVCPB(IDV))*CATD
    1    +CYL(X/2.,IDV)*(X/DVCPB(IDV)))
 332 TD=TD+FLPOV(IFL)*XN*(((1.+((FLRPPO(IFL)-1.)/2.))*(1.-1./FLNCO(IFL)
    1))*CATD+(.5+((FLRPPO(IFL)-1.)/2.)
    1    *(.5+1./FLRPTO(IFL))
    1                                               +(1./FLRPTO(IFL)))*TATD)
     GO TO 1004
 141 TD=TD+FLPOV(IFL)*((2.*XN-1.)*CYL( ((FLNCP(IFL)/FLNBP(IFL))+FLNCO
    1(IFL))/2.,IDV)+(XN*((FLRPPO(IFL)-1.)/2.)*(1.-1./FLNCO(IFL))*CYL((
    2FLNCO(IFL)/2.),IDV))
    1    + ((FLRPTO(IFL)-1.)/2.)*(0.5 +1./ LRPTO(IFL))   +(0.5   +
    2(1./FLRPTO(IFL)))*XN*TATD   )
     IF (FLRPTO(IFL).GT.1.)   TD=TD+.5*FLPOV(IFL)*XN*TATD
    1 *     ((FLRPPO(IFL)-1.)/2.)
     IF (TYP.EQ.B) GO TO 351
     IF(SU.EQ.U) GO TO 144
     TD=TD-FLPOV(IFL)*(TNCO-1.)*(CYL((FLNCP(IFL)/TNCO),IDV)-CATD1)
     GO TO 1004
 351 IF (IDEV.EQ.IDVI) GO TO 352
     IF ( SU.EQ.S) GO TO 353
     TD=TD-(XN-1.)*FLPOV(IFL)*(1.-1./FLNCP(IFL))*(CYL(FLNCO(IFL)/2.,
    1        IDV)-CATD1)
     GO TO 1004
 353 TD=TD-TNCO*FLPOV(IFL)*(CYL(FLNCP(IFL)/TNCO,IDV)-CATD1)
     GO TO 1004
 352  TD=TD-(XN-1.)*FLPOV(IFL)*(CYL((FLNCP(IFL)+FLNCO(IFL))/2.,IDV)
    1        -CATD1)
     GO TO 1004
 144 TD=TD-(XN-1.)*(1.-1./FLNCP(IFL))*FLPOV(IFL)
    1    *(CYL((FLNCO(IFL)/2.),IDV)-CATD1)
1004 IF ((QU.NE.I).AND.(QU.NE.IC)) GO TO 20
     IF (FLPOV(IFL).EQ.0.) GO TO 301
     IF (FLBLPT(IFL).LE.1.) GO TO 302
     IF (QU.EQ.IC) GO TO 303
     TD=TD+XN*TATD*((1.-FLPOV(IFL))*(    ((FLBLPT(IFL)-1.)/2.)*
    1    (2.*CEIL(1./FLBLPT(IFL))+1./FLBLPT(IFL))+5.5+1./FLRPTO(IFL))
    2    +FLPOV(IFL)*(5.5+1./FLRPTO(IFL)))
     GO TO 307
 303 TD=TD+XN*TATD*((1.-FLPOV(IFL))*(    ((FLBLPT(IFL)-1.)/2.)*
    1    (3.*(CEIL(1./FLBLPT(IFL)))+1./FLBLPT(IFL))+9.5
    1                                                    +1./FLRPTO(IFL))
    2    +FLPOV(IFL)*(9.5+1./FLRPTO(IFL)))
```

```
      GO TO 307
  302 IF (QO.EQ.IC) GO TO 304
      TD=TD+XN*TATD*((1.-FLPOV(IFL))*(CEIL(1./FLBLPT(IFL))+1./FLRPTO
     1   (IFL)+5.5)+FLPOV(IFL)*(3.5+2.*CEIL(1./FLRPTO(IFL))))
      GO TO 307
  304 TD=TD+XN*TATD*((1.-FLPOV(IFL))*(2*CEIL(1./FLBLPT(IFL))+
     1   CEIL(1./FLRPTO(IFL))+1./FLRPTO(IFL)+8.5)+FLPOV(IFL)*
     2(5.5  +CEIL(1./FLRPTO(IFL))*4.))
      GO TO 307
  301 IF (FLBLPT(IFL).LE.1.) GO TO 305
      IF (QO.EQ.IC) GO TO 306
      TD=TD+XN*TATD*(    ((FLBLPT(IFL)-1.)/2.)*(2,*CEIL(1./FLBLPT(IFL))
     1    +1./FLBLPT(IFL))+5.5+1./FLRPTO(IFL))
      GO TO 307
  306 TD=TD+XN*TATD*(    ((FLBLPT(IFL)-1.)/2.)*(3.*CEIL(1./FLBLPT(IFL))
     1    +1./FLBLPT(IFL))+9.5+1./FLRPTO(IFL))
      GO TO 307
  305 IF (QO.EQ.IC) GO TO 308
      TD=TD+XN*TATD*(CEIL(1./FLBLPT(IFL))+1./FLRPTO(IFL)+4.5)
      GO TO 307
  308 TD=TD+XN*TATD*(2*CEIL(1./FLBLPT(IFL))+CEIL(1./FLRPTO(IFL))
     1  +1./FLRPTO(IFL)+8.5)
  307 IF (FLMI(IFL).NE.CC) GO TO 370
      IF (QO.EQ.IC) GO TO 371
      TD=TD-2.*XN*TATD
      GO TO 370
  371 TD=TD-3.*XN*TATD
      GO TO 370
  370 IF (FLAM(IFL).EQ.IB) GO TO 312
      IF (IDEV.EQ.IDVI) GO TO 310
      TD=TD+XN*(1.-1./FLINCI(IFL))*CATIN
      GO TO 311
  310 TD=TD+XN*2*CYL((FLNCP(IFL)+FLNCI(IFL))/2.,IDEV)
  311 IF(FLPOV(IFL).EQ.0.) GO TO 20
      TD=TD+XN*FLPOV(IFL)*(1.-1./FLNCO(IFL))*(FLRPPO(IFL)-1.)
     1    *CYL(FLNCO(IFL)/2.,IDVI)
      GO TO 1006
  312 IF (IDEV.EQ.IDVI) GO TO 313
      TD=TD+XN*CYL((FLNCP(IFL)/(2.*FLNBP(IFL))+FLNCO(IFL)/2.),IDEV)
     1    *2.+(1.-1./FLNCI(IFL))*CATIN
      GO TO 314
  313 TD=TD+XN*(CATIF+CYL(FLNCP(IFL)+((FLNCO(IFL)+FLNCI(IFL))/2.),IDEV)
     1    +CATIF)
  314 IF(FLPOV(IFL).EQ.0.) GO TO 20
      TD=TD+XN*FLPOV(IFL)*(1.-1./FLNCO(IFL))*(FLRPPO(IFL)-1.)*
     1     CYL(FLNCO(IFL)/2.,IDEV)
      IF (SO.EQ.S) GO TO 315
      TD=TD+(XN-1.)*(1.-1./FLNCP(IFL))*CATD1
      GO TO 1006
  315 TD=TD+(TNCO-1.)*CATD1
 1006 IF (FLAM(IFL).EQ.IB ) GO TO 20
      IF (FLNB(IFL).EQ.1. ) GO TO 20
      TO=TO +(2.*XN-1.) * FLPOV(IFL)*(1.-1./FLNBO(IFL))*BATO
      IF (SE.EQ.S ) GO TO 1007
      TO = TO - ( XN-1.)*(1.-1.  /FLNBP(IFL))*(1.- FLPOV(IFL))*BATO
      GO TO 1008
 1007 TO = TO - BATO*(1.  - FLPOV(IFL))*(TNBO - 1.)
 1008 TO = TO+XN* ((1.-FLPOV(IFL))*2. *BATO       )
      GO TO 20
   21 IF (OO.EQ.IC) GO TO 196
      IF (FLBLPT(IFL).LE.1.) GO TO 197
      TO=TO+XN*TATO*(CEIL(1./FLBLPT(IFL))+((FLBLPT(IFL)-1.)/2.)*(CEIL(
     11./FLBLPT(IFL))*2.+(1./FLBLPT(IFL)))                   +4.)
      GO TO 20
  197 TO=TO+XN*TATO*(CEIL(1./FLBLPT(IFL))+1./FLRPTO(IFL)+(CEIL(1./FLBLPT
     1(IFL))    +(1./FLBLPT(IFL)))+4.)
      GO TO 20
  196 IF (FLBLPT(IFL).LE.1.) GO TO 198
      TO=TO+XN*TATO*(2.*CEIL(1./FLBLPT(IFL))+((FLBLPT(IFL)-1.)/2.)*(
     1CEIL(1./FLBLPT(IFL))*3.+(1./FLBLPT(IFL)))+3.)
      GO TO 20
  198 TO=TO+XN*TATO*(2.+CEIL(1./FLBLPT(IFL))+CEIL(1./FLRPTO(IFL))+
     1(1./FLRPTO(IFL))+7.+(CEIL(1./FLBLPT(IFL))-(1. /FLBLPT(IFL)) ))
   20 TEMP=TO-TEMP1-TEMP2
  226  FORMAT (14HOOVERFLOW TIME        )
      TT=TT+TO+XN*TP
      ISAM=TT
      IF ((OO.EQ.IC).OR.(OO.EQ.OI)) GO TO 50 '
  192 IF (OO.EQ.O) GO TO 50
      IF (OO.EQ.CC) GO TO 199
      TT=TT+CN*TATO*((1.-FLPOV(IFL))*CEIL(1./FLBLPT(IFL))
     1  + FLPOV(IFL)*CEIL(1./FLRPTO(IFL)))
      ISAM=TT
      GO TO 50
  199 TT=TT+CN*TATO*((1.-FLPOV(IFL))*2.*CEIL(1./FLBLPT(IFL))
     1 +FLPOV(IFL)*2.*CEIL(1./FLRPTO(IFL)))
      ISAM=TT
      IF(PRINT.EQ.NO) GO TO 1113
      WRITE(01,2001) ISAM
 2001 FORMAT(30X,F20.8//)
 1113 CONTINUE
      GO TO 50
C      *****
C      *****
C      BISAM =BASIC INDEXED SEQUENTIAL METHOD
C      *****
C      *****
  209 IF (( FLMI(IFL).NE.M).AND.(FLMI(IFL).NE.MC)) GO TO 171
      IF (IDEV.EQ.IDVI) GO TO 172
      IF (FLMI(IFL).EQ.M) GO TO 173
      TI=CATI+(XN-1.)*CYL((FLINCI(IFL)-1.)/2.,IDVI)
      TI=TI+1.*TATI*XN
      GO TO 175
  173 TI=CATI+(2.*XN-1.)*CYL((FLINCI(IFL)-1.)/2.,IDVI)
```

```
      TI=TI+2.*XN*TATI
      GO TO 175
  172 IF (FLMI(IFL).EQ.M) GO TO 174
      TI=CATD+(XN-1.)*CATIF
      TI=TI+1. *XN*TATD
      GO TO 175
  174 TI=CATD+(XN-1.)*CATIF+XN*CYL((FLINCI(IFL)-1.)/2.,IDVI)
      TI=TI+2.*XN*TAID
      GO TO 175
  171 IF (FLMI(IFL).EQ.CC) GO TO 175
      IF (IDEV.EQ.IDVI) GO TO 176
      TI=CATI+(XN-1.)*CATIN+FLOOR(FLINCI(IFL)/2.) *CATI1
      TI=TI+(     FLINTI(IFL)/2.)*XN*TATI
      GO TO 175
  176 TI=CATD+(XN-1.)*CATIF+FLOOR(FLINCI(IFL)/2.)*CATD1
      TI=TI+CEIL(FLINTI(IFL)/2.)*XN*TATI
  175 WRITE (6,7111)  TI, TO
      IF (SO.EQ.S) GO TO 177
      IF (FLMI(IFL).EQ.CC) GO TO 355
      IF (IDEV.EQ.IDVI) GO TO 178
  355 TI=TI+XN*(1.-1/FLNC(IFL))*CATF
      GO TO 181
  178 TI=TI+XN*CATIF
      GO TO 181
  177  IF(FLMI(IFL).EQ.CC) GO TO 356
      IF (IDEV.EQ.IDVI) GO TO 179
  356 IF (TNCQ.GE.FLNB(IFL)) GO TO 180
      TI=TI+TNCQ*CATD
      GO TO 181
  180 TI=TI+FLNB(IFL)*CATD+(TNCQ-FLNB(IFL))*CYL(FLNC(IFL)/TNCQ,IDEV)
      GO TO 181
  179 TI=TI+XN*CATIF
  181 WRITE (6,7111)  TI, TO
      TI=TI+XN*1. *TATD
      IF ( FLBLPT(IFL).GT.1) GO TO 4025
      TD=XN*(1.-FLPOV(IFL))*(0.5+1./FLBLT (IFL))*TATD
      GO TO 191
 4025 TD =XN*(1.-FLPOV(IFL))*(0.5+(FLBLPT(IFL)-1.)/(2.*FLBLPT(IFL))
     1    +1./FLBLT (IFL) )           *TATD
      GO TO 191
    1 ISAM=QISAM(IFL, XN, TYP, SO,OU, TP,CN,BUNO)
      GO TO 59
   50 WRITE (6,7111)  TI, TO
   59 ISAM=ISAM/1000.
      RETURN
      END
```

```
C       QISAMF
C
C
C
C       QISAM INDEX SEQUENTIAL ACCESS METHOD
C       ********
C       ********
C       QUEUED  ISAM
C       ********
        REAL FUNCTION QISAM(IFL,XN,TYP,SO,QU,TP,CN,BUNO)
C       IFL   =   PTR TO FILE
C       XN   =   NO. RECORDS TO BE RETRIEVED
C       TYP   =   Q   IF QISAM   I.E.   XN  CONSECUTIVE RECORDS
C             =   S   IF SISAM   XN RECORDS INDEPENDENTLY BY KEY
C       SO    =   S   IF SISAM KEYS SORTED
C             =   U   IF SISAM KEYS UNSORTED
C       QU    =   I   IF INSERT TYPE UPDATE
C             =   C   IF MODIFY TYPE UPDATE
C             =   Q   IF QUERY
C       TP    =   PROCESSING TIME  PER RECORD (MS)
C       CN    =   NO. RECORDS COMPLETELY QUALIFYING
C       BUNO = NO. OF BUFFER USED
C
C
C       BUFFERING OPTION - SYSTEM PARAMETER
C       RT RECORD TRACK ONLY
C       TX + TRACK INDEX
C       CI + CYLINDER INDEX
C       MI + MASTER INDEX
C       CEIL(X)   X REAL  'CEILING FUNCTION' - SMALLEST INTEGER GREATER
C                  THAN OR EQUAL TO X
        DATA BO/2HRT/
        DIMENSION FLNCT(20)
        EQUIVALENCE (FLNCT(1),FLTNCT(1))
        DATA IC,CC/2HIC,2HCC/
         REAL MC,IC
        REAL MC
        DATA MC/2HMC/
C       **************************************************
C       DEVICE TABLE
        DIMENSION DVTAB(30,20)
        EQUIVALENCE (DVTAB(1,1),DVNO(1))
        DATA MAXDV,MAXADV/30,20/
C       **************************************************
        COMMON /DV/ NDV,
C                  *INPUT
     1             DVNO                    (30),
     1             DVBPT                   (30),
     1             DVOHBL                  (30),
     1             DVTAT                   (30),
     1             DVTPC                   (30),
     1             DVCAT                   (30),
     1          DVCPB                   (30),
     1          DVBAT                   (30),
     1          DVTART                  (30),
     1          DVTOT                   (30),
     1          DVKGAP         (30),
     1          DVOFAC         (30),
     1           DVCATI             (30),
     1           DVCATL             (30),
C               *COMPUTED AT INPUT TIME
     1          DVETT                   (30)
C
      INTEGER
     1          DVNO                    ,
     1          DVETT
C     ******************************************************
C     FILE TABLE
      DIMENSION FLTAB(20,50)
      EQUIVALENCE (FLTAB(1,1),FLNO(1))
      DATA MAXFL,MAXAFL/20,50/
C     ******************************************************
      COMMON /FL/ NFL,
C               *INPUT
     1          FLNO                    (20),
     1          FLDEV                   (20),
     1          FLIDVN                  (20),
     1          FLBIF                   (20),
     1          FLAM                    (20),
     1          FLTYP                   (20),
     1          FLRPB                   (20),
     1          FLPOV                   (20),
     1          FLOBO                   (20),
     1          FLPPA                   (20),
     1          FLPOF                   (20),
     1          FLOBB                   (20),
     1          FLISZ         (20),
     1          FLMI                    (20)
      COMMON   /FL/
C               *COMPUTED
     1          FLRPPO                  (20),
     1          FLRPPT                  (20),
     1          FLRPTP                  (20),
     1          FLBLPT                  (20),
     1          FLRPTO                  (20),
     1          FLBLT                   (20),
     1          FLNTP                   (20),
     1          FLNCP                   (20),
     1          FLNBP                   (20),
     1          FLTNTI                  (20),
     1          FLTNCI                  (20),
     1          FLTNR                   (20),
     1          FLRSIZ                  (20),
     1          FLNTO                   (20)
```

```
      COMMON /FL/
     1            FLNCH                    (20),
     1            FLNBO                    (20),
     1            FLNB                     (20),
     1            FLNC                     (20),
     1            FLINIP                   (20),
     1            FLIDEV                   (20),
     1            FLIDVI                   (20),
     1            FLETI                    (20),
     1            FLSEG                    (20),
     1            FLIFD                    (20)
      INTEGER
     1            FLNO                     ,
     1            FLDEV                    ,
     1            FLIDVN                   ,
     1            FLIDEV                   ,
     1            FLETI                    ,
     1            FLIDVI                   ,
     1            FLSEG                    ,
     1            FLIFD
      COMMON/PCTRL/PRINT
      DATA NO/2HNO/
      REAL NO
      DATA RI,IX,CI/2HRI,2HII,2HCI/
      REAL IB,I,IE
      REAL MI,M,IS
      DATA IB,I,IE/2HIB,2H I,2HIE/
      DATA MI,M/2HMI,2H M/
      DATA C/2H C/
      DATA B/1HB/,
      DATA BN/6H      /
      DATA S, J,O,IS,SI,OI/1HS,2H O,1HO,2HIS,2HSI,2H I/
      INTEGER OT
      DATA OT/6/
      IF(PRINT.EQ.NO) GO TO 1111
      WRITE(OT,100) IFL,XN,TYP,SU,QU,TP,CN,BOND
 100  FORMAT(20X,9HCALL ISAM,I10,F10.0,9XA2,9XA1,8XA2,F9.3,F9.0,F9.0//)
1111  CONTINUE
      TI=0.
      TO=0.
      IDV   = FLIDEV(IFL)
      IDEV = FLIDEV(IFL)
      IDVI  =  FLIDVI(IFL)
      TATO= DVTAT(IDV)
      CATO= DVCAT(IDV)
      BATO = DVBAT(IDV)
      TATI= DVTAT(IDVI)
      CATI= DVCAT(IDVI)
      CATO1=DVCAT1(IDV)
      CATOL=DVCATL(IDV)
      CATI1=DVCAT1(IDVI)
      CATIL=DVCATL(IDVI)
      CATF=CYL((FLNCP(IFL)/2.),IDV)
      CATIN=CYL((FLNCI(IFL)/(2.               )),IDVI)
      CATIF=CYL((FLNCI(IFL)+FLNCP(IFL))/2.,IDV)
      TNR  = FLTNR(IFL)
      TNBLQ = FLOOR(QUAL(CN/XN ,FLRPB(IFL),
     1 (XN/FLRPB(IFL))*(1.-FLPOV(IFL)))        )
      IF(FLRPPO(IFL).EQ.1.) TNUCQ = XN*FLPOV(IFL)
      IF(FLRPPO(IFL).NE.1.) TNUCQ = QUAL(XN/TNR,FLRPPO(IFL),FLTNIP(IFL))
      TNTQ = QUAL(XN/TNR,FLRPPT(IFL),FLTNIP(IFL))
      IF(FLAM(IFL).EQ.IE) S1 = FLNIP(IFL)
      IF(FLAM(IFL).NE.IE) S1 = DVTPC(IDEV) -1.
      TNCQ = QUAL(TNTQ/FLTNIP(IFL),S1              ,FLNCP(IFL))
      IF(FLAM(IFL).EQ.IB)
     1   S2= FLOOR(DVCPB(IDEV)/(1.+ (FLPOV(IFL)*FLRPTP(IFL)/((1.-FLPOV
     1       (IFL))*FLRPTO(IFL)*FLPOF(IFL)))))
      IF(FLAM(IFL).NE.IB) S2 = DVCPB(IDEV)
      TNBQ = QUAL(TNCQ/FLNCP(IFL),S2             ,FLNBP(IFL))
      TNTQI= QUAL(TNCQ/FLNCP(IFL),FLNCP(IFL)/FLTNTI(IFL),FLTNTI(IFL))
      TNCQI= QUAL(TNTQI/FLTNTI(IFL),DVTPC(IDVI),FLTNCI(IFL))
      IF(PRINT.EQ.NO) GO TO 1112
      WRITE (OT,1999)
 1999 FORMAT(30X5HTNBLQ,9X,5HTNUCQ,9X,4HTNTQ,10X,4HTNCQ,10X,4HTNBQ,
     1  10X,4HTNQI,10X,5HTNCQI)
      WRITE(OT,2000) TNBLQ,TNUCQ,TNTQ,TNCQ,TNBQ,TNTQI,TNCQI
 2000 FORMAT (26X,7F14.4)
 1112 CONTINUE
    1 IF ((QU.NE.I).AND.(QU.NE.IC)) GO TO 35
      IF(FLNC(IFL).GT.1.) GO TO 161
      TD=CATD+BATD+(1.+FLTNIP(IFL))*TATD
      IF(QU.NE.IC) GO TO 23
      TD=TD+(1.+FLTNIP(IFL))*TATD
   23 WRITE (6,7111)  TI, TD
      QISAM=TD
      GO TO 50
  161 IF(IDEV.EQ.IDVI) GO TO 162
      TD=CATI+(FLTNCI(IFL)-1.)*CATII
      TD=TD+FLNCP(IFL)*3.*TATI
      TD=TD+FLNB(IFL)*CATD+(FLNCP(IFL)-FLNB(IFL))*CATDI+
     1    FLNB(IFL)*BATD+FLNC(IFL)*(DVTPC(IDEV)-1.)*3.*TATD+
     2    FLNB(IFL)*0.5*TATD
      IF(QU.NE.IC) GO TO 23
      TD=TD+FLNC(IFL)*DVTPC(IDEV)*TATD*2.+FLNCP(IFL)*TATD
      GO TO 23
  162 IF(FLNB(IFL).LE.1.) GO TO 163
      TD=(DVCPB(IDV)-FLNCI(IFL))*CATD*2.+(1.-(DVCPB(IFL)-FLNCI(IFL))/
     1    FLNCP(IFL))*FLNCI(IFL)*CATII+FLNCP(IFL)*3.*TATD
      WRITE (6,7111)  TI, TD
 7111 FORMAT ( 10X,F15.5, 10X,F15.5)
      TD=TD+FLNB(IFL)*CATD+FLNB(IFL)*BATD+FLNCP(IFL)*(DVTPC(IFL)-1.)
     1    *3.*TATD+FLNB(IFL)*0.5*TATD
      WRITE (6,7111)  TI, TD
      IF(QU.NE.IC) GO TO 23
      TD=TD+FLNC(IFL)*2.*DVTPC(IFL)*TATD+FLNCP(IFL)*TATD
      GO TO 23
  163 TD=CATI+(FLNCP(IFL)-1.)*2.*CATIF+FLNCP(IFL)*3.*TATD
      WRITE (6,7111)  TI, TD
      TD=TD+FLNB(IFL)*BATD+FLNC(IFL)*(DVTPC(IDV)-1.)*TATD*3.+.5*TATD
      WRITE (6,7111)  TI, TD
      IF(DD.NE.IC) GO TO 23
      TD=TD+FLNC(IFL)*DVTPC(IDV)*2.*TATD+FLNCP(IFL)*TATD
      GO TO 23
   35 IF((DD.EQ.CC).OR.(DD.EQ.C)) GO TO 300
      IF(FLAM(IFL).NE.IE) GO TO 200
  209 TD=BATD*(ONE(XN*FLNBP(IFL)/FLTNR(IFL)))+CATD1*(ONE(XN*FLNCP(IFL)/
     1FLTNR(IFL)))
  202 IF(FLBLPT(IFL).GT.1.)GO TO 203
      FN=1.
      TD=TD+XN*(1.-FLPOV(IFL))*(CEIL(1./FLBLPT(IFL)))*TATD/FLRPB(IFL)
     1 +XN*(1.-FLPOV(IFL))*1.5*TATD/FLRPB(IFL)
      WRITE (6,7111) TI,TD
      IF((FLBLT(IFL)-FLBLPT(IFL)).GT.0.1) GO TO 204
      TD=TD+XN*FLPOV(IFL)*CEIL(1./FLBLPT(IFL))*TATD
     1+XN*FLPOV(FIL)*TATD
      GO TO 205
  204 TD=TD+XN*FLPOV(IFL)*CEIL(1./FLBLPT(IFL))*TATD
      GOTO  205
  203 IF((BUND/2.).GT.FLBLPT(IFL)) GO TO 206
      FN=FLOOR(BUND/2.)
  207 TD=TD+(XN*(1.-FLPOV(IFL))/(FLRPB(IFL)*FN)+XN*(1.-FLPOV(IFL))*
     11.5/(FLRPB(IFL)*FLBLPT(IFL)))*TATD+XN*FLPOV(IFL)*TATD
      GO TO 205
  206 FN=FLBLPT(IFL)
      GO TO 207
  200 IF (FLPOV(IFL).EQ.0.) GO TO 209
      IF (FLAM(IFL).EQ.IB) GO TO 210
      TD=BATD*(ONE(XN*FLNBP(IFL)/FLTNR(IFL)))
     1                                      +CATD1*(ONE(XN*FLNCP(IFL)/
     1FLTNR(IFL)))
      WRITE (6,7111) TI,TD
      X=FLNCO(IFL)/DVCPB(IDV)
      Y=FLOOR(FLNCO(IFL)/DVCPB(IDV))
      WRITE (6,7111) X,Y
      IF (X.GT.1.) GO TO 211
      TD=TD+ FLOOR(FLPOV(IFL)*XN)*(1.-1./FLNCO(IFL))*CYL(FLNCO(IFL)/2.
     1 ,  IDEV)
      WRITE (6,7111) TI,TD
      GO TO 212
  211 TD=TD+(FLOOR(FLPOV(IFL)*XN)*((1.-1./FLNCO(IFL))*(1./X)))*
     1     (Y*CATF+(CATD1+((X-Y)/2.)*(CATDL-CATD1))*(X-Y))
      WRITE (6,7111) TI,TD
  212 IF (FLBLPT(IFL).GT.1.) GO TO 202
      IF(FLBLPT(IFL).EQ.1.) GO TO 250
  251 IF (FLPOV(IFL).GT.0.5) GO TO 214
      TD=TD+(XN-1.)*0.5*TATD*FLPOV(IFL)
      WRITE (6,7111) TI,TD
      GO TO 202
  250 IF (FLRPB(IFL).EQ.1.) GO TO 202
      GO TO 251
  214 TD=TD+XN*0.5*(1.-FLPOV(IFL))*TATD
      GO TO 202
  213 IF ((FLPOV(IFL)*FLRPB(IFL)*FLBLPT(IFL)/(1.-FLPOV(IFL))).GT.1.)
     1 GO TO 215
      TD=TD+XN*FLPOV(IFL)*0.5*TATD
      WRITE (6,7111) TI,TD
      GO TO 202
  215 TD=TD+XN*(1.-FLPOV(IFL))*0.5*TATD/(FLRPB(IFL)*FLBLPT(IFL))
      WRITE (6,7111) TI,TD
      GO TO 202
  210 TD=TD+BATD*(ONE(XN*FLNBP(IFL)/FLTNR(IFL)))
      WRITE (6,7111) TI,TD
      IF (FLNCP(IFL).GT.(DVCPB(IDEV)-FLNCU(IFL))) GO TO 242
      PCAT=CYL((FLNCP(IFL)+FLNCU(IFL))/2.,IDEV)
      UCAT=CYL(               FLNCU(IFL) /2.,IDEV)
       WRITE (6,7111) PCAT,UCAT
      GO TO 243
  242 X=FLNCP(IFL)/(DVCPB(IDEV)-FLNCU(IFL))
      Y=FLOOR(X)
      PCAT=(1./X)*(Y*CATD+(
     1                     CATD1+((X-Y)/2.)*(CATDL-CATD1)))*(X-Y))
      UCAT=(1./X)*(Y*CYL(FLNCU(IFL)/2.,IDV)     +(X-Y)
     1       *CYL((X-Y)*FLNCU(IFL)/2.,IDV)   )
      WRITE (6,7111) PCAT,UCAT
      WRITE (6,7111) X,Y
  243 BADT=FLRPB(IFL)*FLBLPT(IFL)*FLPOV(IFL)/(1.-FLPOV(IFL))
      IF(BADT.GE.1.) GO TO 244
      WRITE (6,7111) X,Y
      TD=TD+2.*XN*FLPOV(IFL)*PCAT+CATD1*(1.-FLRPB(IFL)*FLBLPT(IFL)*
     1      FLPOV(IFL)/(1.-FLPOV(IFL)))*ONE(XN*FLNCP(IFL)/FLTNR(IFL))
      WRITE (6,7111) X,Y
      WRITE (6,7111) TI,TD
      GO TO 212
  244 TD=TD +XN*FLINTP(IFL)*2.*PCAT/FLTNR(IFL)
     1                               +(FLRPPU(IFL)-1.)*UCAT
     1      *XN*FLINTP(IFL)/FLTNR(IFL)
      WRITE (6,7111) TI,TD
      GO TO 212
  205 WRITE (6,7111) TI,TD
  299 GO TO 165
C     **************************************
C     **************************************
C     **************************************
C      MODIFY RECORDS
C     **************************************
C     **************************************
  300 IF ((BUNO/2.).GT.FLBLPT(IFL)) GO TO 301
      IF(((2.*FLBLPT(IFL)/BUNO)-FLOOR(2.*FLBLPT(IFL)/BUNO)).GT.0.)
     1                GO TO 302
 301  CON=1.
      GO TO 303
 302  CON=2.
 303  IF (CN.NE.XN) GO TO 400
      IF (FLAM(IFL).NE.IE) GO TO 304
 315  TD=TD+BATD*(ONE(XN*FLNBP(IFL)/FLINR(IFL)))+
     1   CON*CATD1*(ONE(XN*FLNCP(IFL)/FLINR(IFL)))
 321  FN=FLOOR(BOND/2.)
      IF (FLBLPT(IFL).LE.1.) GO TO 305
      IF (FN.GE.FLBLPT(IFL)) GO TO 306
      IF(((FLBLPT(IFL)/FN)-(FLOOR(FLBLPT(IFL)/FN))).NE.0.) GO TO 307
      TD=TD+((FLBLPT(IFL)/FN+2.)*((XN*(1.-FLPOV(IFL)))/(FLRPB(IFL)
     1       *FLBLPT(IFL)))-1.)*TATD
      IF (GO.NE.CC) GO TO 308
      TD=TD+(FLBLPT(IFL)/FN)*(XN*(1.-FLPOV(IFL))/(FLRPB(IFL)*FLBLPT(IFL)
     1 )     )*TATD
 308  TD=TD+XN*FLPOV(IFL)*2.*TATD
      IF (GO.NE.CC) GO TO 399
      TD=TD+XN*FLPOV(IFL)*TATD
      GO TO 399
 306  TD=TD+(3.* XN*(1.-FLPOV(IFL))/(FLRPB(IFL)*FLBLPT(IFL))-1.)*TATD
      IF (GO.NE.CC) GO TO 308
      TD=TD+(3.*XN*(1.-FLPOV(IFL))/(FLRPB(IFL)*FLBLPT(IFL))-1.)*TATD
      GO TO 308
 307  AI=1.
 314  CM=AI*FLBLPT(IFL)
      IF(((CM/FN)-FLOOR(CM/FN)).EQ.0.) GO TO 309
      AI=AI+1.
      GO TO 314
 309  TIC=CM/FLBLPT(IFL)
      BIC=CM/FN
      IF (FN.GT.(FLBLPT(IFL)/2.)) GO TO 310
      TD=TD+((BIC+2*TIC-1.)*(1./TIC)*(XN*(1.-FLPOV(IFL))/
     1       (FLRPB(IFL)*FLBLPT(IFL)))-1.)*TATD
      GO TO 311
 310  TD=TD+(2.*(BIC+TIC)*(1./TIC)*(XN*(1.-FLPOV(IFL))/
     1       (FLRPB(IFL)*FLBLPT(IFL)))-1.)*TATD
 311  IF (GO.NE.CC) GO TO 308
      TD=TD+BIC*(1./TIC)*(XN*(1.-FLPOV(IFL)
     1  )                                      /(FLRPB(IFL)*FLBLPT(IFL)))*TATD
      GO TO 308
 305  FN=1.
      TD=TD+XN*(1.-FLPOV(IFL))*(2.*CEIL(1./FLBLPT(IFL))+1.)*TATD
     1/FLRPB(IFL)
      IF (GO.EQ.CC) TD=TD+CEIL(1./FLBLPT(IFL))*TATD*XN*(1.-FLPOV(IFL))
     1/FLRPB(IFL)
      IF ((FLBLT(IFL)-FLBLPT(IFL)).GT.0.1) GO TO 312
      TD=TD+XN*FLPOV(IFL)*2.*
     1                            CEIL(1./FLBLPT(IFL))*TATD+XN*FLPOV(IFL)
     1     *TATD
      GO TO 313
```

```
312 TD=TD+XN*FLPOV(IFL)*2.*CEIL(1./FLBLPT(IFL))*TATD
313 IF (QU.EQ.CC) TD=TD+XN*FLPOV(IFL)*CEIL(1./FLBLPT(IFL))*TATD
399 FK=2.
    GO TO,299
304 IF (FLPOV(IFL).EQ.0.) GO TO 315
    IF (FLAM(IFL).EQ.IB)  GO TO 316
    TD=TD+BATD*(ONE(XN*FLNBP(IFL)/FLTNR(IFL)))+
   1       CATD1*CON*(ONE(XN*FLNCP(IFL)/FLTNR(IFL)))
    X=FLNCO(IFL)/DVCPB(IDEV)
    Y=FLOOR(FLNCO(IFL)/DVCPB(IDEV))
    IF (X.GT.1.) GO TO 317
    TD=TD+ FLOOR(FLOOR(IFL)*XN)*( 1.-1./FLNCO(IFL))*2.
   1      *CYL((FLNCO(IFL)/2.),IDEV)
    GO TO 318
317 TD=TD+FLOOR(FLPOV(IFL)*XN)* (1.-1./FLNCO(IFL))*2.
   1   *(Y*CATF+( CATD1+((X-Y)/2.)*(CATDL-CATD1))*(X-Y))
318 IF(((FLBLPT(IFL)).GT.1.).OR.((FLBLT(IFL)-FLBLPT(IFL)).GT.0.1))
   1      GO TO 319
    IF (FLPOV(IFL).GE.0.5) GO TO 320
    TD=TD+2.*
   1          XN*0.5*FLPOV(IFL)*2.*TATD
    GO TO 321
320 TD=TD+(2.*XN-1.)*0.5*TATD
    GO TO 321
319 IF((FLPOV(IFL)*FLBLPT(IFL)*FLRPB(IFL) /(1.-FLPOV(IFL))).GE.1.)
   1   GO TO 322
    TD=TD+XN*FLPOV(IFL)*0.5*TATD*2.
    GO TO 321
322 TD=TD+XN*(1.-FLPOV(IFL))*0.5*TATD*2./(FLRPB(IFL)*FLBLPT(IFL))
    GO TO 321
316 TD=TD+BATD*(ONE(XN*FLNBP(IFL)/FLTNR(IFL)))+
   1   CON*CATD1*(ONE(XN*FLNCP(IFL)/FLTNR(IFL)))
    IF ((XN*FLPOV(IFL)).GT.(XN*FLTNTP(IFL)/FLTNR(IFL))) GO TO 323
    AC=XN*FLPOV(IFL)
    W=0.
    GO TO 324
323 AC=XN*FLTNTP(IFL)/FLTNR(IFL)
    W=XN*FLPOV(IFL)-AC
324 IF((FLNCP(IFL)).GT.(DVCPB(IDEV)-FLNCO(IFL))) GO TO 325
    TD=TD+(4.*ONE(AC)-1.)*CYL((FLNCP(IFL)+FLNCO(IFL))/2.,IDEV)
    GO TO 326
325 X=(FLNCP(IFL)/(DVCPB(IDEV)-FLNCO(IFL)))
    Y=FLOOR( FLNCP(IFL)/(DVCPB(IFL)-FLNCO(IFL)))
    TD=TD+(4.*ONE(A)-1.)*(1./X)*(Y*CATF+CATD1+((X-Y)/2.)*(CATDL-CATD1)
   1       )
326 TD=TD+ W* (1.-1./FLNCO(IFL))*CYL(FLNCO(IFL)/2.,IDEV)*2.
    GO TO 318
400 FN=FLOOR(BUNO/2.)
    ATEM=XN* FLNCP(IFL)/FLTNR(IFL)
    WRITE (6,7111) ATEM
    WRITE (6,7111) TI,TD
    IF (FN.GT.FLBLPT(IFL)) FN=FLBLPT(IFL)
```

```
      INBUQ=QUAL(CN/XN,FN*FLRPB(IFL), (XN/FLTNR(IFL))*
     1       FLTNIP(IFL)*FLBLPT(IFL)/FN)
      INBTQ=QUAL(CN/XN,FLRPPI(IFL),(XN/FLTNR(IFL))*FLTNTP(IFL))
      WRITE (6,7111) INBUQ,INBTQ
      IF (FLAM(IFL).NE.IE) GO TO 404
415   TD=TD+BATD*(ONE(XN*FLNBP(IFL)/FLTNR(IFL)))
     1   +CATD1*((ONE(XN*FLNCP(IFL)/FLTNR(IFL)))
     1           +(( TNBTQ/ATEM )           /CEIL(FLTNTP(IFL)/ FLNCP(IFL))))
      WRITE (6,7111) TI,TD
      IF (FLPOV(IFL).GT.0.) GO TO 430
431   IF (FLBLPT(IFL).LE.1.) GO TO 405
421   IF (FN.GE.FLBLPT(IFL)) GO TO 406
      IF (((FLBLPT(IFL)/FN)-FLOOR(FLBLPT(IFL)/FN))
     1                                                   .NE.0.) GO TO 407
      WRITE (6,7111) TI,TD
      TD=TD+((FLBLPT(IFL)/FN+1.)*(XN*(1.-FLPOV(IFL))/(FLRPB(IFL)*
     1       FLBLPT(IFL)))+FN*TNBUQ/FLBLPT(IFL)-1.)*TATD
      WRITE (6,7111) TI,TD
      IF (QU.EQ.CC) TD=TD+INBUQ*TATD
408   TD=TD+(XN*FLPOV(IFL)+CN*FLPOV(IFL))*TATD
      WRITE (6,7111) TI,TD
      IF (QU.EQ.CC) TD=TD+CN*FLPOV(IFL)*TATD
      GO TO 299
406   TD=TD+(( 2.*XN*(1.-FLPOV(IFL))
     1                                      /(FLRPB(IFL)*FLBLPT(IFL))-1.)
     1            +INBTQ)*TATD
      IF (QU.EQ.CC) TD=TD+INBTQ*TATD
      GO TO 408
430   IF (FLRPPO(IFL).GT.1.) GO TO 432
      FR=FLRPB(IFL)*FLBLPT(IFL)*FLPOV(IFL)/(1.-FLPOV(IFL))
      GO TO 433
432   FR=1./FLRPPO(IFL)
433   TD=TD+CATD1*FR*CN*FLPOV(IFL)*TNBTQ/ATEM      /
     1  CEIL(FLTNTP(IFL)/ FLNCP(IFL))
      GO TO 431
407   AI=1.
414   CM=AI*FLBLPT(IFL)
      IF (((CM/FN)-FLOOR(CM/FN)).EQ.0.) GO TO 409
      AI=AI+1.
      GO TO 414
409   TIC=CM/FLBLPT(IFL)
      BIC=CM/FN
      WRITE (6,7111) TIC,BIC
      IF (FN.GT.(FLBLPT(IFL)/2.)) GO TO 410
      WRITE (6,7111) TI,TD
      TD=TD+((BIC+2*TIC)*XN*(1.-FLPOV(IFL))/(TIC*FLRPB(IFL)*FLBLPT(IFL))
     1       -1.)*TATD+INBUQ*TATD/BIC
      GO TO 411
410   TD=TD + ((BIC+2*TIC-1.)*XN*(1.-FLPOV(IFL))
     1                                                  /(TIC*FLRPB(IFL)*
     1           FLBLPT(IFL))-1.+INBUQ*(1.+1./BIC))*TATD
      WRITE (6,7111) TI,TD
```

```
411 IF (QU.EQ.CC) TD=TD+TNBUQ*TATD
    GO TO 408
405 FN=1.
    TD=TD+XN*(1.-FLPOV(IFL))*(CEIL(1./FLBLPT(IFL))+1.)*TATD/FLRPB(IFL)
   1    +    (1.-FLPOV(IFL))*CEIL(1./FLBLPT(IFL))*TATD*TNBUQ
    WRITE (6,7111) TI,TD
    IF (QU.EQ.CC) TD=TD+    (1.-FLPOV(IFL))*CEIL(1./FLBLPT(IFL))*TATD*
   1TNBUQ
    IF((FLBLT(IFL)-FLBLPT(IFL)).GT.0.1) GO TO 412
    TD=TD+(XN*FLPOV(IFL)*CEIL(1./FLBLPT(IFL))+XN*FLPOV(IFL)
   1        +CN*FLPOV(IFL)*CEIL(1./FLBLPT(IFL)))*TATD
    WRITE (6,7111) TI,TD
    GO TO 413
412 TD=
   1  TD+(XN+CN)*FLPOV(IFL)*CEIL(1./FLBLPT(IFL))*TATD
    WRITE (6,7111) TI,TD
413 IF (QU.EQ.CC) TD=TD+CN*FLPOV(IFL)*CEIL(1./FLBLPT(IFL))*TATD
    GO TO 299
404 IF (FLPOV(IFL).EQ.0.) GO TO 415
    IF (FLAM(IFL).EQ.IB) GO TO 416
    WRITE (6,7111) TI,TD
    TD=TD+BATD*(ONE(XN*FLNBP(IFL)/FLTNR(IFL)))+
   1        CATD1*(ONE(XN*FLNCP(IFL)/FLTNR(IFL)))
    X=FLNCO(IFL)/DVCPB(IDEV)
    Y=FLOOR(FLNCO(IFL)/DVCPB(IDEV))
    IF (X.GT.1.) GO TO 417
    TD=TD+(XN+CN)*FLPOV(IFL)*(1.-1./FLNCO(IFL))*CYL(FLNCO(IFL)/2.,IDV)
    WRITE (6,7111) TI,TD
    GO TO 418
417 TD=TD+(XN+CN)*FLPOV(IFL)*(1.-1./FLNCO(IFL))*(1./X)
   1    *(Y*CATD+(
   1                CATD1+(X-Y)*(CATDL-CATD1)/2.)*(X-Y))
418 IF (FLBLPT(IFL).GT.1.) GO TO 202
    IF(FLBLPT(IFL).EQ.1.) GO TO 450
451 IF (FLPOV(IFL).GT.0.5) GO TO 440
    TD=TD+2.*(XN+CN)*FLPOV(IFL)*0.5*TATD
    WRITE (6,7111) TI,TD
    GO TO 431
450 IF (FLRPB(IFL).EQ.1.) GO TO 202
    GO TO 451
440 TD=TD+(XN+CN-1.)*0.5*TATD*2.
    GO TO 431
419 IF ((FLPOV(IFL)*FLBLPT(IFL)*FLRPB(IFL)/(1.-FLPOV(IFL))).GE.1.)
   1        GO TO 441
    TD=TD+XN*FLPOV(IFL)*0.5*TATD+CN*FLPOV(IFL)*0.5*TATD
    WRITE (6,7111) TI,TD
    GO TO 431
441 TD=TD+XN*(1.-FLPOV(IFL))*0.5*TATD/(FLRPB(IFL)*FLBLPT(IFL))
   1        +TNBTQ*0.5*TATD*(1./FLBLPT(IFL))
    GO TO 431
416 TD=TD+BATD*(ONE(XN*FLNBP(IFL)/FLTNR(IFL)))
    IF (FLNCP(IFL).GT.(DVCPB(IDEV)-FLNCO(IFL))) GO TO 442
```

```
      PCAT=CYL((FLNCP(IFL)+FLNCU(IFL))/2.,IDEV)
      UCAT=CYL(                FLNCU(IFL) /2.,IDEV)
       WRITE (6,7111) PCAT,UCAT
      GO TO 443
  442 X=FLNCP(IFL)/(DVCPB(IDEV)-FLNCU(IFL))
      Y=FLOOR(X)
      WRITE (6,7111) X,Y
      PCAT=(1./X)*(Y*CATD+(
     1                          CATD1+((X-Y)/2.)*(CATDL-CATD1))*(X-Y))
      UCAT=(1./X)*(Y*CYL(FLNCU(IFL)/2.,IDV)      +(X-Y)
     1        *CYL((X-Y)*FLNCU(IFL)/2.,IDV)    )
      WRITE (6,7111) PCAT,UCAT
  443 BADT=FLRPB(IFL)*FLBLPT(IFL)*FLPOV(IFL)/(1.-FLPOV(IFL))
      IF(BADT.GE.1.) GO TO 444
      WRITE (6,7111) TI,TO
      TO=TO+2.*(XN+CN)*FLPOV(IFL)*PCAT+CATD1*(ONE(XN*FLNCP(IFL)/
     1       FLTNR(IFL))*(1.-FLRPB(IFL)*FLBLPT(IFL)*FLPOV(IFL)/
     1      (1.-FLPOV(IFL)))+TNBTO*(FN/ONE(FLBLPT(IFL))*2.*(1./
     1      (DVTPC(IDV)-1.))))
      WRITE (6,7111) TI,TO
      GO TO 418
  444  TO=TO+((XN*2.*FLTNTP(IFL)/FLTNR(IFL))+TNBTO*
     1       (FN/ONE(FLBLPT(IFL))))*PCAT+((FLRPPU(IFL)-1.)
     1    *XN*FLTNTP(IFL)/FLTNR(IFL)
     1      +CN*FLPOV(IFL)*(1.-1./FLRPPU(IFL)))*UCAT
     1      +(CN*FLPOV(IFL)*2.*PCAT/FLRPPU(IFL))
      WRITE (6,7111) TI,TO
       GO TO 418
  165 WRITE (6,7111)  TI, TO
      IF (FLPOV(IFL).EQ.0.) GO TO 30
      WRITE (6,9121)
 9121 FORMAT(1X,1H1)
      IF (FLAM(IFL).EQ.IB.OR.FLAM(IFL).EQ.IE) GO TO 30
      WRITE (6,9122)
 9122 FORMAT(1X,1H2)
      IF(FLNB(IFL).EQ.1.0) GO TO 30
      WRITE (6,9123)
 9123 FORMAT(1X,1H3)
      TO = TO + BATD*(2.0+(FLRPPU(IFL)-1.)*(1.-1./FLNBU(IFL)))
   30 IF((FLMI(IFL).EQ.M).OR.(FLMI(IFL).EQ.MC)) GO TO 166
      WRITE (6,9124)
 9124 FORMAT(1X,1H4)
      IF (FLMI(IFL).EQ.CC) GO TO 167
      WRITE (6,9125)
 9125 FORMAT(1X,1H5)
      TI=(ONE(FLTNTI(IFL)/2.)+0.5)*TATI+ONE(FLTNCI(FIL)/2.)*CATI1
     1  + 1.5*TATD          +CATI+CATD
      GO TO 168
  167 TI=1.5*TATD       +CATD
      GO TO 168
  166 IF (FLMI(IFL).EQ.MC) GO TO 169
      WRITE (6,9126)
```

```
9126 FORMAT (1X,1H6)
     IF (BO.EQ.MI) GO TO 169
     TI=(1.+(1.-1./FLTNCI(IFL)))*CATIN+2*TATI+1.5*TATO +CATO
     GO TO 168
 169 TI=CATI+1.5*TATI+1.0*TATO    +CATO
 168 WRITE (6,7111)  TI, TO
     QISAM= TO + TI
     IF(PRINT.EQ.NO) GO TO 1114
     WRITE (6,9127)
9127 FORMAT (1X,1H7)
     WRITE(OT,2001) QISAM
2001 FORMAT (30X, E20.8//)
1114 CONTINUE
  50 RETURN
     END
```

SECTION VI

PHASE II
A DATA MANAGEMENT SYSTEM MODEL

P. J. Owens

PHASE II
A DATA MANAGEMENT SYSTEM MODEL

by

P. J. Owens

Information Sciences Department
IBM Research Laboratory
San Jose, California

ABSTRACT: The advent of modern data management systems has raised the need for models of such systems, and for computer programs which embody these models for simulation purposes. Typically, transactions on such systems result in complex patterns of accesses to direct access storage devices. These access patterns are dependent on several characteristics of the data management system, among which are:

(1) The contents of the data base;

(2) The organization and accessibility;

(3) The nature of the request.

Furthermore, once the sequence of requests is determined, the efficiency of the system in satisfying these requests is most dependent (or potentially so) on the hardware configuration itself. Hence, it is desirable to develop models which reflect these dependencies. We think we have taken a step in that direction.

I. INTRODUCTION

The advent of modern data management systems has raised the need for models of such systems, and for computer programs which embody these models for simulation purposes. Typically, transactions on such systems result in complex patterns of accesses to direct access storage devices. These access patterns are dependent on several characteristics of the data management system, among which are:

(1) The contents of the data base;

(2) The organization and accessibility;

(3) The nature of the request.

Furthermore, once the sequence of requests is determined, the efficiency of the system in satisfying these requests is most dependent (or potentially so) on the hardware configuration itself. Hence, it is desirable to develop models which reflect these dependencies. We think we have taken a step in that direction.

The purpose of this section is to describe PHASE II, a model of data management systems which has been implemented as a set of computer programs. Some of the ideas for PHASE II evolved from experiences related to the development of an earlier model, FOREM I, described in (1). The implementation of FOREM I, which used analytic techniques and was very fast, was found, on the other hand, to be deficient in several respects:

(1) Some configurations of data, hardware, and access methods defied analysis;

(2) The introduction of a new parameter increased the complexity of the resulting analysis manyfold;

(3) The analytic programs were difficult to debug and verify;

(4) It was impossible to simulate simultaneous I/O operations.

Therefore, we decided that a program more closely mirroring activity on computer systems in general, and data management systems in particular, should be developed, with the effect of sacrificing run time efficiency for flexibility, generality, and ease of development, modification, and generalization. Insofar as the FOREM I programs are valid, however, they can be adapted for use in the PHASE II system.

Progı ᵐ specifications and design are not stressed in this paper because they are not complete or general and might tend to obscure the model itself. Specifications of how to use the modeling programs and details of program design can be found in section 6 (the Phase II User Guide).

II. OBJECTIVES OF THE MODELING EFFORT

The PHASE II modeling program was designed with several objectives in mind:

(1) To provide a means whereby data bases with known characteristics and transaction sets and/or file activity profiles can be run against variations in hardware configuration, physical arrangement of data on devices, data set organization, and accessing strategy.

(2) To provide a means whereby general studies can be made for issuing guidelines and trade-off curves for data base and retrieval system design; to search out relationships between the characteristics of a data management systems environment; and to identify the most important characteristics of a given subset of characteristics, that is, those to which the performance of the system is most sensitive.

(3) To provide diagnosis of and possible improvements to existing systems by examining resource utilization statistics for I/O bottlenecks.

(4) To allow a modeler desiring to do (1), (2) or (3) to characterize a data management system environment at the required level of detail for those aspects of the system under scrutiny. The modeler will, in turn, be furnished by the modeling programs with the required statistics for evaluating the simulated system.

III. MODELING DATA MANAGEMENT SYSTEMS

It is useful to think about a model of a system in terms of two major aspects of the model:

(A) The static model, which is a description of the logical and physical configurations of the elements involved, and a stimulus to be applied to the model.

(B) The dynamic model, which is a description of how the configuration changes when a given stimulus is applied, and how long it takes.

These two submodels have direct analogues in the programs that implement the model, typically assuming the form of program tables to implement the static model, and executable program statements to implement the dynamic model.

For data management systems, the static model can be thought of as having four major submodels:

(1) Logical description of the data;

(2) Hardware configuration;

(3) The mapping of the data base onto hardware devices;

(4) A description of the transaction to be performed.

The dynamic model has three major submodels:

(5) Identification of those data elements which need to be accessed to complete the transaction;

(6) Locating those elements on the hardware devices;

(7) Accessing them in the desired order.

There exist simple, general models of (1) and (2). However, (3) is difficult to characterize at once succinctly and with generality; as many schemes for providing a data-device map exist as there are data management systems. Now, (4) depends on (1), (5) depends on (4), (6) depends on (5) and (3), and (7) depends on (4).

Because of these complex interdependencies, and an inability to characterize succinctly some of the submodels, two restrictions have been placed on the kind of system to be described by the PHASE II model:

(1) The data must be describable in terms of a hierarchical structure;

(2) The data must be conventionally stored; that is, related fields are stored together in "records," and all records of the same type are, in some sense, stored together.

These restrictions are somewhat vaguely stated here with the intent of conveying the modest design objectives of the model. Their exact meaning is specified in the sections to follow, which define the model in more precise detail.

A subset of this model may, of course, be used to model systems which do not "fit" the PHASE II model, but at a level more primitive than that implied by the above discussion. For example, if a transaction set for a system can be characterized by a sequence of accesses to well-defined locations on the hardware devices, thus bypassing the logical data description and data-device map, the above restrictions would not affect the applicability of this model to the system.

IV. THE PHASE II MODEL

The PHASE II model allows one to characterize and simulate a data management system with respect to eight aspects of such a system:

(1) Data field contents;

(2) Logical structure of the data;

(3) Physical organization of the data;

(4) Data selection criteria;

(5) Data accessing methods;

(6) Accessing strategy;

(7) Hardware;

(8) I/O supervisor.

In the following eight sections, we will discuss each of these aspects and indicate how they are characterized.

1. Data Field Contents

A data field is usually thought of as an item of information about a particular entity; for example, a person's name, a company's assets, etc. A data field's contents can be characterized in the form of a density distribution of its values over all occurrences of the field in the data base. Any given value that the field can take on is assumed to be uniformly distributed throughout all occurrences of the field. The one exception to this is for a "sort" field, in which case the order in which the values are presented in the distribution is the order in which they will appear in the data set (to be defined later) containing the field.

Fields are assumed to be statistically independent of each other and of other system parameters (except for sort fields), hence, data bases involving fields with significant correlational effects will require careful treatment, perhaps in some cases by lumping correlated fields together and treating them as a single field.

2. Logical Structure of the Data

The logical structure of a data base imposes a relational structure on the fields of the data base, and can be thought of as a "user's view" of the data base, as opposed to the "system programmer's view" of the data base. Such a structure has an existence independent of any associations of the data with a specific data management system used to store and access the data. As was stated previously, we confine ourselves to hierarchical data structures described as follows:

Data fields are organized into groups of related fields, or "segments." A segment may have sets of inferior segments related to it, thus inducing a segment hierarchy, or tree structure, on the data.

For example, a personnel file may contain information having the structure depicted in Figure 1. This structure consists of two hierarchical levels. Level 0 contains nonrepeating information about an employee, and level 1 contains two types of segments with recurrent information; namely, a list of positions

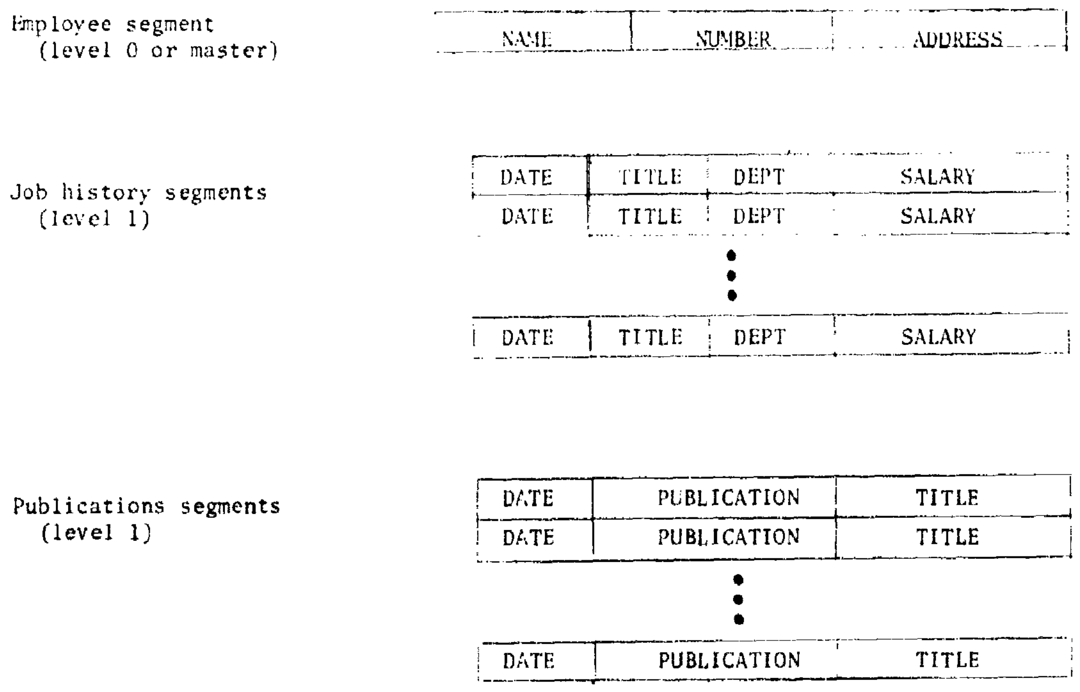

Employee segment (level 0 or master)
NAME
NUMBER
ADDRESS
Job history segments (level 1)
DATE
TITLE
DEPT
SALARY
Publications segments (level 1)
DATE
PUBLICATION
TITLE

FIGURE 1

Hierarchical Data Structure

the employee has held with the company, and a list of his publications. The latter two will, of course, occur different numbers of times for different employees.

A general structure of this type will allow subordination to any level, and as many different segment types at each level as the application requires. The one restriction is that a strict tree structure must be maintained, that is, a segment type may occur at only one level. A given instance of such a structure (in this example, all the information about a particular employee) is called a "logical record," and the collection of all such records (the personnel file) is called a "logical file." The data base may contain several logical files.

3. Physical Organization of the Data

The physical organization of the data is a specification of how the logical files are to be stored on physical devices. This logical-to-physical mapping is carried out in three steps:

(a) Partitioning of logical records into "logical subrecords" to form "data sets."

(b) Assignment of each data set to one or more "elementary files".

(c) Partitioning of elementary files into "extents" on hardware devices.

In the first step, each segment type (that is, all of its occurrences) is assigned to a unique data set, which is defined as a collection of records numbered from 1 to N. Figure 2 demonstrates such a partitioning for the personnel file example cited in the previous section. The "employee segment" and associated "job history: segments are assigned to data set 1, on a one record per employee basis, as illustrated in Figure 3. All "publication" segments are assigned to data set 2 on a one record per publication basis.

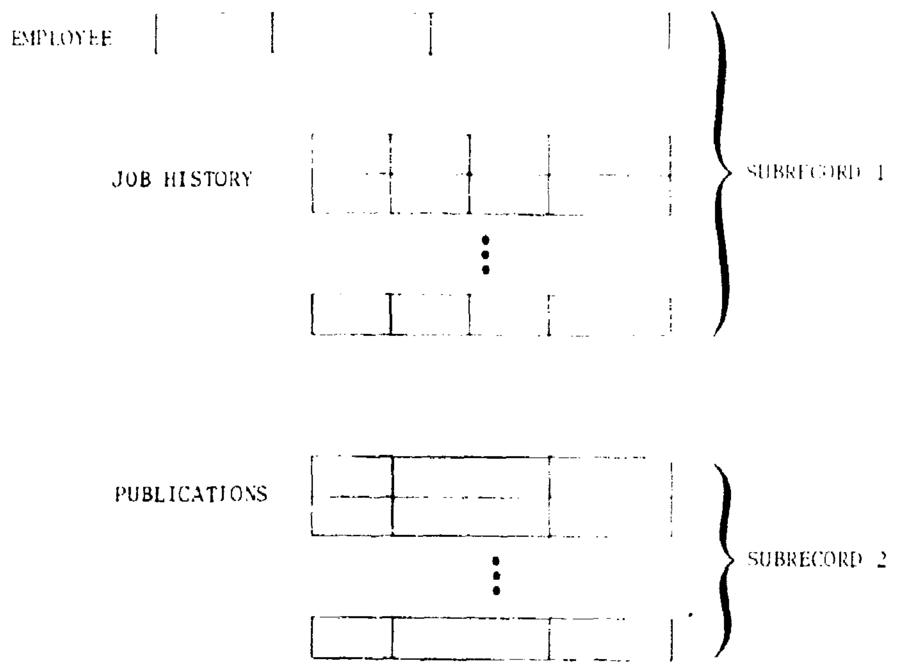
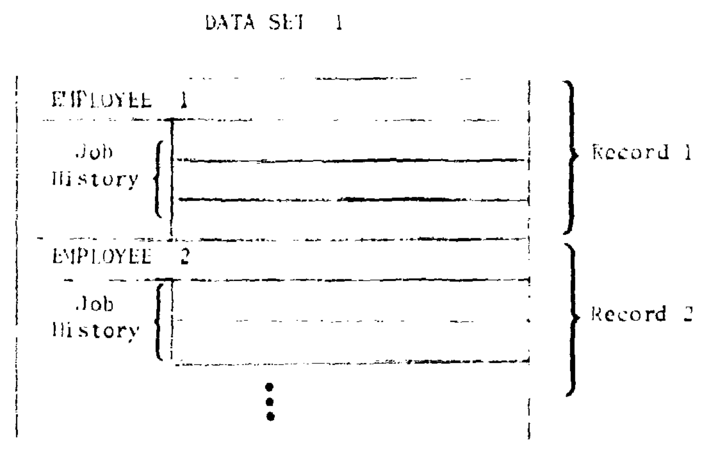


FIGURE 2
Partitioning of Logical Records into Logical Subrecords



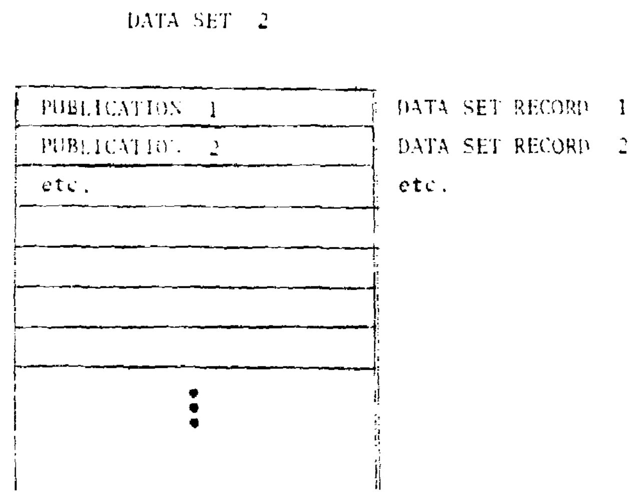


FIGURE 3
Assignment of Logical Subrecords to Data Set Records

There are two restrictions on the way in which the above partitioning can be carried out:

(1) If two segment types have been assigned to a data set, they must have a common ancestor which is also assigned to the data set.

(2) If two segments which are lineally related are assigned to the same data set, segment types occurring between them in the segment hierarchy must also be assigned to that data set.

The serialization of the records of a data set constitutes an important interface between the data accessing methods and the data accessing strategy in that the record number is the primary means of referring to information for retrieval. The serial number also provides the ordering which forms the basis for the notion of sort field.

At this point, we still are dealing with abstract, or logical entities, namely logical records, logical subrecords, data sets, and records. A data set is mapped onto physical devices by means of building blocks called "elementary files." How many elementary files are needed to represent a data set, and what their contents are, depends on how the data in that data set is to be accessed. For instance, if the records of a data set are stored and accessed sequentially, only one elementary file is required. On the other hand, if the records are to be accessed "randomly" by means of indexes, the data itself, the indexes, and other auxiliary files would each constitute an elementary file.

By definition, an elementary file is a collection of information recorded on storage devices, with the following properties:

(1) It may reside on one or more devices of the same type (that is, having the same physical characteristics), and occupy different numbers of cylinders on each device.

(2) It occupies the same number of tracks on each cylinder, except possibly the last cylinder occupied by the file.

(3) Physical record format (that is, record size and blocking characteristics) is the same throughout the file.

Each elementary file is, in turn, partitioned into "extents" and so mapped onto physical devices. Each extent is characterized by naming the device on which the extent resides, the first cylinder to be occupied, and the number of consecutive cylinders occupied.

The elementary file characterization is the main instrument used in locating a given record of a data set (and auxiliary information, such as index records).

4. Data Selection Criteria

Data selection criteria are roughly equivalent to what are commonly referred to as "queries," in the sense that a query usually specifies a set of characteristics which a logical record (or subrecord) must satisfy in order for it to qualify for retrieval or further action.

For example, (going back to the personnel file example) one may wish to retrieve personnel records of all persons who have taken positions in Dept. 25 since 1965. We shall assume that there are, on the average, three job history segments per master segment, that 50% of them have dates later than 1965, and that 20% of them have DEPT = 25. (The above information would be available from the field and logical structure specifications.) Assuming field independence, 10% of the job history segments fully qualify. Further assuming that the segments that thus qualify are uniformly distributed throughout the file, the fraction of people qualifying (that is, those with one or more qualifying job history segments) is:

$$1 - (\text{fraction of records with no qualifying job history segments})$$

$$= 1 - (\text{probability that a job segment does not qualify})^3$$

$$= 1 - (1 - .10)^3 = .27$$

This kind of calculation can be extended to hierarchies of arbitrary depth and complexity; however, the modeler should give careful consideration to the assumptions involved.

A hierarchical model of a data structure introduces a semantic problem into the query specification in that, to avoid ambiguity, a more complicated selection specification is required than would be required for nonhierarchical data. This can best be demonstrated by an example.

Consider a hierarchy consisting of two segment types: superior segment A and inferior segment B. Each B segment has fields x and y. Such a hierarchy is depicted in Figure 4.

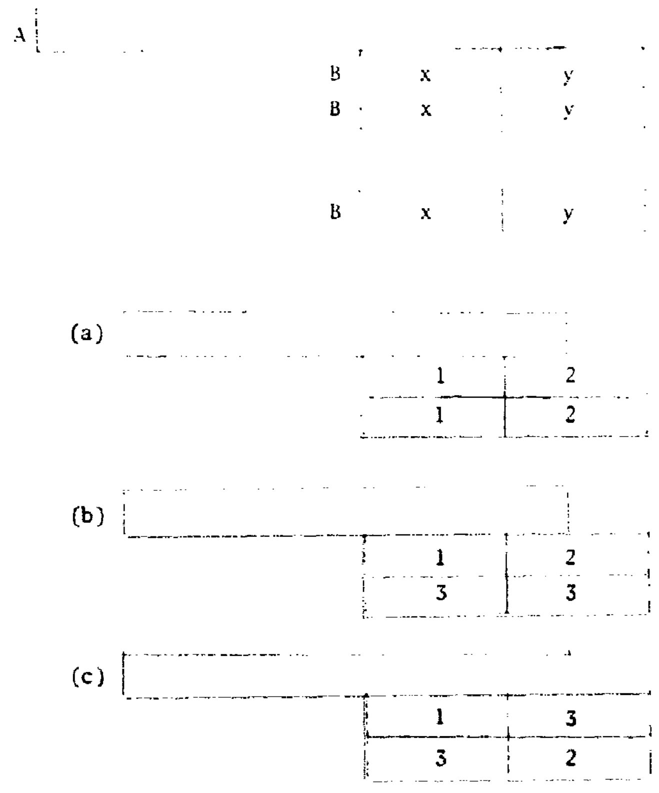


FIGURE 4

A Logical Data Structure and Specific Instances

Consider also specific instances of this structure (a), (b) and (c); also depicted in Figure 4, and the following query:

"Find all segments A such that x = 1 and y = 2."

This query, as it stands, might have several interpretations, as supplied by the following appendages to the query:

(1) "co-occurring in a subordinate segment B at least once."

(2) "anywhere, not necessarily co-occurring."

(3) "for all B segments inferior to A."

Of the specific instances (a), (b) and (c), the qualifying instances for the three interpretations are:

1. (a) and (b)

2. (a), (b), and (c)

3. (a)

Therefore, it is clear that what is needed is a more powerful characterization of a query (or qualification specification) than can be supplied by a simple Boolean expression. Such a characterization can take the form of a statement with the following form:

> "SEG qualifies by criterion LBL if it has QUANT related ELEMENTs that satisfy QUAL"

where the capitalized elements are defined as follows:

LBL - an arbitrarily assigned qualification name or label

SEG	-	name of a segment
ELEMENT	-	a field or segment. An element is "related" to SEG if it is SEG itself, a descendent segment of SEG, an ancestor segment of SEG, or a field in any of these segments.
QUANT	-	a quantifier on ELEMENT
QUAL	-	a qualification criterion on ELEMENT. If ELEMENT is a field name, QUAL will specify a subset of the range of the field. If ELEMENT is a segment name, QUAL will be a reference to a qualification label of a qualification statement on that segment, or some Boolean combination thereof.

LBL	SEG	QUANT	ELEMENT	QUAL
Q	B		x	= 1
R	B		y	= 2
S	A	any	x	= 1
T	A	any	y	= 2
U	A	any	B	Q and R
V	A	any	A	S and T
W	A	all	B	Q and R

FIGURE 5
Resolution of the Ambiguity Problem

In Figure 5, queries (1), (2), and (3) have been expressed unambiguously by qualification statements U, V, and W, respectively.

Following are some examples of how the statements in Figure 5 are interpreted:

Q: "A B segment qualifies by criterion Q if its x field equals 1." (The quantifier is not necessary, since each B segment has exactly one x field.)

U: "An A segment qualifies by criterion U if any of its B segments are qualified by both criteria Q and R."

W: "An A segment qualifies by criterion W if all of its B segments are qualified by both criteria Q and R."

A transaction set on a particular data base may consist of many thousands of queries and updates. Such a set can be characterized by partitioning the transactions into subsets of transactions whose form is the same within subsets, but whose field value qualifiers change from transaction to transaction. Hence, in addition to the form of the query, the modeler would need to supply for each queried field a distribution from which field values to be queried on are to be selected. This again makes certain assumptions about statistical independence which may or may not be well-founded in specific instances. Once the characterization of the transaction sets is made, field values can be selected at random from the distributions, the transaction so defined can be simulated, and this process can be repeated as many times as is required by the modeler.

Each qualification statement defines a list of qualifying records of the data set in which the qualified segment appears. How this list is used in characterizing the accessing of records is described in the next two sections.

5. Data Accessing Methods

Once the modeler has defined the elementary files of a data set, he then needs to specify how a given record is to be accessed in response to a request. That is, he must specify the sequence of accesses to the elementary files of the data set which ultimately result in the retrieval of a requested record. Of course, this dynamic aspect of the retrieval process is intimately tied to the meaning of the elementary files which constitute the data set; in fact, it supplies the meaning.

Each data accessing method represents a different way of retrieving records from data sets. Some of the more common techniques are:

(1) Sequential access: This method consists of sequentially leafing through a data set, record by record, until the requested record is reached. Sequential access is very efficient if one wishes to access all the records of a data set in the order in which they are stored. It allows anticipatory reading and buffering, so that the requestor may not have to wait for I/O to take place before he can process the next record.

(2) Indexed access: This method involves first referencing an index, which can give either the approximate location of the desired record, to which the user must go and search sequentially until he finds it; or the location of a lower level index, which, in turn, specifies either the above mentioned, or another level of index.

(3) Direct access: This method allows the user to go directly to the record desired in that the record is requested by location rather than by name.

Each of these methods has many variations, each of which can result in drastic variations in operating characteristics; thus, it is almost impossible to provide a brief characterization of an accessing method. It can, however, be characterized by a computer program which simulates the operation of such a method. Hopefully, the interfaces between such a program and the simulation system environment with which it interacts can be straightforward and simple, so that a modeler wishing to simulate his own accessing technique (and familiar with the language in which the model is implemented) would need only a minimal amount of instruction.

In the present model, programs are provided to simulate well known accessing methods such as the IBM OS/360 Sequential Access and Indexed Sequential Access methods.

6. Accessing Strategy

Let us review the picture of the model that has been presented so far. We have described the logical description of the data, and its physical realization in the form of data sets. The data accessing methods provide us with a way of accessing a single record from a data set. The qualification specifications supply us with lists of records which need to be accessed to fulfill requests for information from the system. The final step is to provide a way of describing the order in which the records on the lists are to be accessed; that is, a description of the interrelation of data sets and data set accesses in fulfilling a query. This description is analogous to the lower level description of an accessing method, which describes the interrelationships of the elementary files of a data set in fulfilling a request for a single record of the data set.

In general, the accessing strategy specification allows the modeler to describe:

(1) Lists of records to be accessed from the data sets involved.

(2) The accessing method to be used in accessing a given set of records from a data set.

(3) The order in which the accesses are to occur.

For example, a modeler may wish to read records 1, 3, 5, ... from sequential data set A, records 200, 400, 600, ... from indexed data set B, and merge these records onto sequential data set C. This example uses all of the above three elements: the lists (1, 3, 5, ... and 200, 400, 600, ...), the accessing methods (sequential and indexed), and the ordering (read from A, write to C, read from B, write to C, ...).

Three basic specifications are used to characterize such strategies (in the form of a simple procedural language):

(1) The LIST specification, which defines a set of records to be accessed. Such a list can be a literal list of record numbers, a sequential or skip sequential list, a random list taken from a given distribution, or a random or sequential list of records which qualify on the basis of a qualification specification. This last mentioned option provides the only link between the qualification specification and the accessing strategy.

(2) The ACCESS OP specification, which identifies the accessing method to be used, the data set to be accessed, and the record to be accessed. The

last of these is obtained from a specified list, and removed from the list, so that on the next execution of the statement, the "next" record on the list will be accessed from the data set.

(3) The SYNC specification, which allows one to specify a random interleaving of operations on two or more data sets. Such a specification is necessary to describe merge-type operations.

7. Hardware

A simple hardware configuration is assumed for the purposes of this model, as depicted in Figure 6, namely: one CPU with one or more channels, each having one or more control units, each of which has one or more devices attached. The modeler may specify that a control unit is switchable between two or more channels. In such a case, a control unit may be logically attached to at most one channel at any given moment, and will remain attached to that channel until the "current" request is satisfied.

The above summarizes the topology of the hardware elements. Each device (e.g., a disk drive, drum, etc.) in turn is characterized by assigning it to a device class, all of which have the same physical characteristics; for example, all 2311 disk drives form a device class.

Direct access devices are characterized by such parameters as rotational period, number of tracks per cylinder, cylinder access time (which may be a function of two variable: current cylinder and sought cylinder), maximum record size, gap factors, and so on.

8. Input/Output Supervisor

The function of the I/O supervisor is to accept requests, marshal them through various queues, and see them through the completion. This component of the model (like the accessing methods) is characterized by a program, which, usually, is called upon by an accessing method, and, in turn, interfaces with the hardware in its current state.

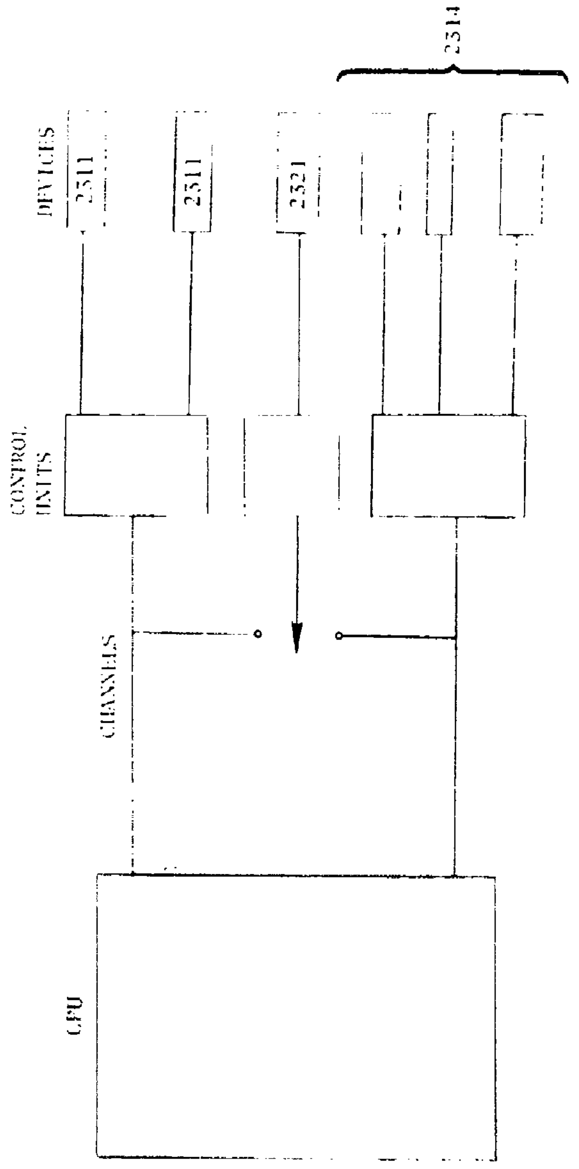


FIGURE 6
Hardware Model

The I/O supervisor is implemented as a very simple event-driven queueing model, in which the stations are devices, channels, and the CPU, and the events are begin and end seek, begin and end transmit, and begin and end CPU processing. It essentially assumes all the functions of an operating system (other than data management); hence, the name is something of a misnomer. However, I/O events are assumed to be the predominant concern of this model.

Briefly (see Figure 7), an I/O request to the I/O supervisor is specified in the form "request I/O from device N1, cylinder N2, track location X1, transmit time X2, operation type X3 (read, write, etc.)." This request is placed on a seek queue for the requested device, and when channel, control unit, and device are free, the seek is initiated, thus tying up the device. At end-of-seek, the request is placed in a transmit queue for the appropriate channel, and when channel and control units are free, and the requested track position comes under the head, transmission takes place. This operation ties up device, control unit, and channel. Finally, the requesting program is signalled that its request has been satisfied. The I/O supervisor maintains current hardware status (arm position, channel busy, etc.) and advances the clock.

An accessing method may also issue a WAIT for a particular request, and it may issue a PROCESS for a given time T, which is effectively a guarantee that the program will issue no more requests during time T.

This model allows the modeler to simulate the effects of device and channel separation on data sets simultaneously being accessed. If such detail is not required, simpler I/O supervisor programs can be substituted, or in fact, it may simply be ignored by accessing modules which compute their own timing characteristics.

V. CONCLUSION

We have attempted in this section to describe a model of a certain type of data management system. The restrictions placed on the type of system which "fits" the model are sufficiently severe to make the resulting model relatively simple (that is, relative to a full-blown model), yet general enough to model a wide range of possible systems.

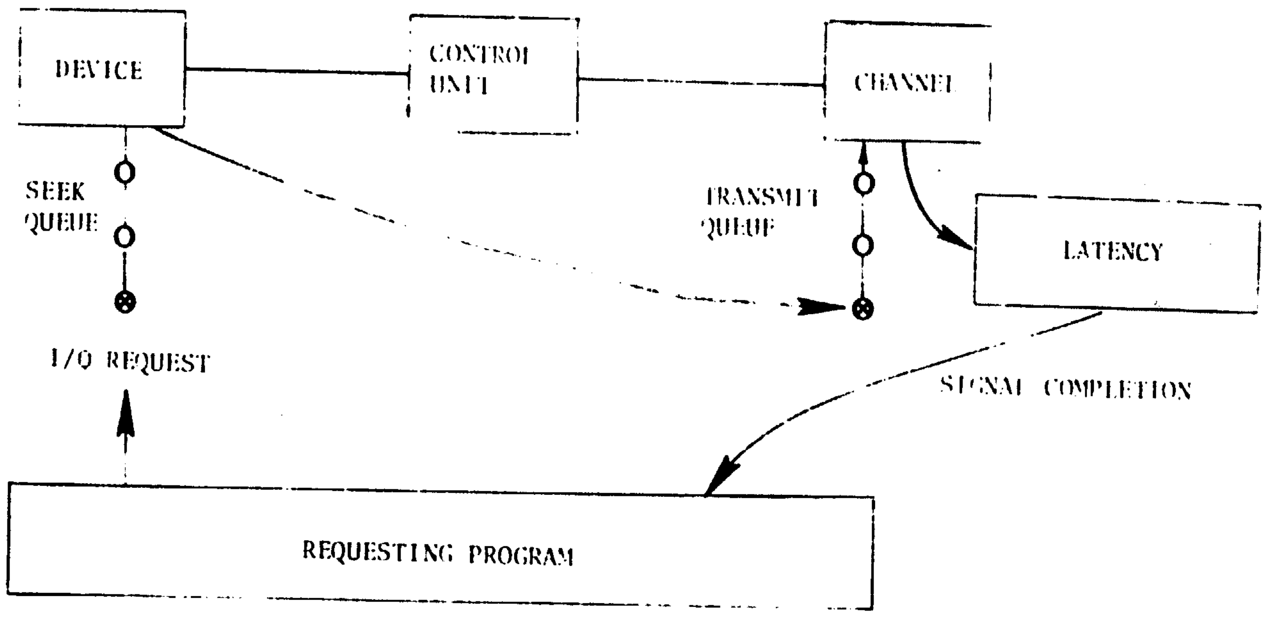


FIGURE 7
I/O Supervisor

Furthermore, we believe that the design of model and programs will allow future generalizations to areas not touched, and that the modeling effort will develop ways of thinking about such systems which will lead to more general models.

VI A NOTE ON THE IMPLEMENTATION OF PHASE II

Presently, the model exists as a program at one of its specified levels of "completion." Elements of all the above described aspects have been included at this point. The program consists of about 8,000 lines of FORTRAN code, and occupies a load module of 215K bytes, including tables.

The speed at which the modeling program operates is roughly proportional to the number of accesses it is required to simulate, with a rule of thumb being 500 microseconds (on the mod 91) per hardware access simulated. The output of the model currently consists of timings of interest to the modeler. Future levels of the model envision a statistics gathering capability which will (at the user's option) gather information on wait times, queue lengths, hardware activity, and so on.

VII. DOCUMENTATION AND DELIVERY OF THE PHASE II SYSTEM

The documentation for the Phase II system consists of this section and section 6 (the Phase II User Guide). The system itself will be delivered on a magnetic tape containing six files whose contents are described in section 11 of the User Guide. They contain, among other things, source and object modules of the system, and the User Guide. Accompanying the distribution tape is a computer output listing which gives example OS/360 Job Control Language for installing and maintaining the system. The JCL as distributed will probably need to be modified somewhat to conform to installation conventions.

VIII. BIBLIOGRAPHY

1. M. E. Senko, V. Y. Lum, P. J. Owens, "A File Organization Evaluation Model (FOREM)," IFIP Congress 68, Edinburgh, August 1968.
2. M. E. Senko, et al.,"Formatted File Organization Techniques - Final Report" RADC Contract No. AF 30(602)-4088, May 16, 1968.

SECTION VII

THE PHASE II DATA MANAGEMENT SIMULATION SYSTEM (Level 2)

USER GUIDE

P. J. Owens

THE PHASE II DATA MANAGEMENT SIMULATION SYSTEM

(LEVEL 2)

USER GUIDE

P. J. OWENS

IBM RESEARCH, SAN JOSE

0.0 TABLE OF CONTENTS

1.0 INTRODUCTION TO PHASE II

2.0 CONTROL CARDS

3.0 INPUT TABLES

3.1 HARDWARE
3.2 DEVICE TYPE
3.3 SEGMENT
3.4 DATASET
3.5 QUALIFICATION
3.6 PROCEDURE
3.7 LIST
3.8 TABLE
3.9 DISTRIBUTION
3.10 TABLE SUMMARY

4.0 EXAMPLE MODEL SPECIFICATIONS

5.0 ERROR MESSAGES

6.0 MODEL SIZE LIMITATIONS

7.0 ACCESS METHODS

7.1 ADDING ACCESS METHODS TO PHASE II
7.2 DIRECT ACCESS METHOD
7.3 SEQUENTIAL ACCESS METHOD
7.4 INDEX-SEQUENTIAL ACCESS METHOD

8.0 I/O SUPERVISOR

9.0 ORGANIZATION OF THE PHASE II SYSTEM

9.1 TABLES
9.2 FLOW OF CONTROL
9.3 MODULES AND ENTRY POINTS

10.0 INTERNAL PARAMETER DEFINITIONS

11.0 SYSTEM DISTRIBUTION AND MAINTENANCE

12.0 MATHEMATICAL MODELS

12.1 QUALIFICATION CALCULATION
12.2 ISAM OVERFLOW CHAIN LENGTH DISTRIBUTION

1.0 INTRODUCTION TO PHASE II

THE PHASE II DATA MANAGEMENT SIMULATION SYSTEM IS AN ATTEMPT TO PROVIDE A SIMULATION MODEL OF COMPUTER SYSTEMS WHICH ARE DATABASE ORIENTED, I/O BOUND, AND WHOSE SIGNIFICANT EVENTS OCCUR ON A MILLISECOND TIME SCALE. IT IS ORIENTED TOWARD DATABASES WHICH REPRESENT HIERARCHICALLY ORGANIZED DATA STORED IN A MORE-OR-LESS CONVENTIONAL FASHION ON DIRECT ACCESS DEVICES.

WE ASSUME A SINGLE-CPU CONFIGURATION WITH A SINGLE-TASKING OPERATING SYSTEM IN A BATCH ENVIRONMENT.

BRIEFLY, PHASE II ALLOWS A USER TO SPECIFY A HARDWARE CONFIG-URATION, A DATABASE DESCRIPTION, A MAPPING OF THE DATABASE ONTO THE HARDWARE DEVICES, A SET OF DATA QUALIFICATION CRITERIA, AND A PROCEDURE FOR CARRYING OUT THE DATABASE TRANSACTIONS. CONCOM-ITANT FACILITIES, SUCH AS TABLES, DISTRIBUTIONS, AND LISTS, ARE ALSO PROVIDED.

THE PRINCIPAL OUTPUTS OF THE MODEL ARE TIMINGS OF THE PROCESSES OF INTEREST TO THE MODELER. FUTURE VERSIONS OF THE MODEL WILL ALSO PROVIDE SUMMARIES OF VARIOUS OTHER STATISTICS TO BE GATHERED BY THE MODEL, SUCH AS CHANNEL UTILIZATION, AVERAGE WAIT TIMES, AND SO ON.

IT IS SUGGESTED THAT ONE NOT FAMILIAR WITH THE MODEL FIRST TURN HIS ATTENTION TO SECTION 4, WHICH BY THE USE OF SIMPLE BUT INCREASINGLY COMPLEX EXAMPLES CONVEYS THE FLAVOR OF THE MODEL. THESE EXAMPLES SHOULD BE STUDIED IN CONJUNCTION WITH THE APPROPRIATE TABLE DESCRIPTIONS IN SECTION 3, WHICH ALSO SUPPLY DETAILED SPECIFICATIONS FOR REFERENCE PURPOSES.

A MORE COMPLETE TREATMENT OF THE MODEL ON WHICH PHASE II IS BASED IS GIVEN IN "PHASE II - A DATA MANAGEMENT SYSTEM MODEL", BY THE AUTHOR, AND IS A COMPANION DOCUMENT TO THIS ONE.

2.0 CONTROL CARDS

A MODEL SPECIFICATION CONSISTS OF "CONTROL CARDS" AND "INPUT TABLE CARDS". THIS SECTION DEFINES THE TYPES AND MEANINGS OF THE CONTROL CARDS.

EACH CONTROL CARD INDICATES TO THE SYSTEM A TYPE OF PROCESSING TO BE PERFORMED; FOR EXAMPLE, READ HARDWARE TABLES, EXECUTE PROCEDURE, AND SO ON. IN ANY RUN, ONLY ONE OCCURRENCE OF CONTROL CARDS 1-11 MAY APPEAR, WITH THE EXCEPTION OF "PROCEDURE" AND "EXECUTE", WHICH MAY BE RE-SPECIFIED TO PERMIT EXECUTION OF SEVERAL PROCEDURES IN ONE RUN.

INPUT TABLES MAY BE SPECIFIED IN ANY ORDER.

CONTROL CARDS HAVE THE FOLLOWING FORMAT:

```
CCL   1             15   20   25
      *KEYWORD             P    N
```

WHERE:

KEYWORD	SPECIFIES THE TYPE OF PROCESSING. WHEN KEYWORD SPECIFIES THAT A TABLE IS TO BE READ IN, P AND N ARE INTERPRETED AS FOLLOWS:
P	=BLANK - PRINT TABLE AS READ IN ON STANDARD OUTPUT
	=NON-BLANK - DO NOT PRINT
N	FORTRAN LOGICAL FILE FROM WHICH THE TABLE IS TO BE READ. IF BLANK OR ZERO, STANDARD INPUT IS ASSUMED, IN WHICH CASE THE APPROPRIATE INPUT TABLE CARDS IMMEDIATELY FOLLOW THE CONTROL CARD IN THE INPUT STREAM.

THE FOLLOWING CONTROL CARDS ARE DEFINED:

1. *HARDWARE

   READ HARDWARE CONFIGURATIONS AND PHYSICAL CHARACTERISTICS

2. *DEVICE CLASS

   READ PARAMETERS DEFINING DEVICE CLASSES. FOUR DEVICE CLASSES ARE BUILT INTO THE SYSTEM, AND MAY BE REFERRED TO BY NAME: 2314,2321,2311,2302.

3. *DATASETS

   READ DATASET CONFIGURATIONS AND PARAMETERS

4. *SEGMENTS

READ SEGMENT CONFIGURATIONS AND PARAMETERS

5. *QUALIFICATION

READ QUALIFICATION SPECIFICATIONS

6. *PROCEDURE

READ PROCEDURE

7. *LISTS

READ LIST SPECIFICATIONS

8. *DISTRIBUTIONS

READ DISTRIBUTION SPECIFICATIONS

9. *TABLES

READ TABLE SPECIFICATIONS

10. *EXECUTE

EXECUTE PROCEDURE. IF NO PROCEDURE IS DEFINED, THE PROGRAM WILL BRANCH TO SUBROUTINE "ALTPR", TO BE SUPPLIED BY THE USER.

11. *END

END OF PROCESSING

THE FOLLOWING THREE CONTROL CARDS ARE FOR USE AS DEBUGGING AIDS, BUT ARE INCLUDED FOR THE SAKE OF COMPLETENESS:

12. *PRINT

PRINT EACH TABLE AFTER IT IS INPUT AND AFTER IT IS INTERPRETED. SUCCEEDING OCCURRENCES OF THIS CARD WILL ALTERNATELY TERMINATE AND RE-INITIATE SUCH PRINTING.

13. *DUMP

DUMP EACH TABLE AFTER IT IS INPUT AND AFTER IT IS INTERPRETED. SUCCEEDING OCCURRENCES OF THIS CARD WILL ALTERNATELY TERMINATE AND RE-INITIATE SUCH DUMPING.

14. *TRACE

TRACE ROUTINES AS SPECIFIED BY A USER SUPPLIED "BLOCK DATA" PROGRAM.

3.0 INPUT TABLES

INPUT TABLE ENTRIES HAVE THE FOLLOWING FORMAT:

```
COL   1 - 4      5      6   -  15    16   -  71   72
      LABEL             KEYWORD      PARAMETERS   CONTINUATION
```

WITH THE FOLLOWING CONVENTIONS:

1. PARAMETERS MAY BE SEPARATED BY COMMAS AND/OR ONE OR MORE BLANKS

2. TWO CONSECUTIVE COMMAS INDICATE THE ABSENCE OF A PARAMETER

3. A NON-BLANK IN COL. 72 MEANS THAT THE PARAMETER LIST CONTINUES ON THE NEXT CARD

4. IF A CARD ENDS WITH A COMMA, CONTINUATION ON THE NEXT CARD IS ASSUMED

5. COMMAS MUST NOT BE CODED FOR ABSENT TRAILING PARAMETERS

6. THERE IS A LIMIT OF 115 CHARACTERS FOR A PARAMETER LIST. (A PARAMETER OF LENGTH N CHARACTERS COUNTS AS N+1 CHARACTERS, AND IF THE PARAMETER LIST STARTS AFTER COLUMN 16, THE LEADING BLANKS ARE ALSO COUNTED)

7. THE LABEL MUST START IN COL. 1

8. THE KEYWORD MUST START IN OR AFTER COL. 6, AND END IN OR BEFORE COL. 15

9. VALUES TO BE INPUT MAY BE REAL, INTEGER, OR ALPHAMERIC, AS IMPLIED BY THE MEANING OF THE PARAMETER. THE FORMS WHICH THESE VALUES MAY ASSUME ARE:

INTEGER	A STRING OF CONTIGUOUS DIGITS, WHICH MAY BE PREFIXED BY A MINUS SIGN
REAL	LIKE INTEGER, EXCEPT A DECIMAL POINT MAY APPEAR
ALPHA	A STRING OF NOT MORE THAN FOUR CONTIGUOUS CHARACTERS. COMMAS, PARENTHESES, OR BLANKS MAY NOT APPEAR IN AN ALPHA PARAMETER.

THE FOLLOWING IS A DESCRIPTION OF INPUT CONVENTIONS AND DEFINITIONS OF INPUT PARAMETERS FOR ALL TABLES. EACH DESCRIPTION CONTAINS A DISCUSSION OF THE INPUT TABLE, FOLLOWED BY A "PROTOTYPE" EXAMPLE OF THE TABLE, FOLLOWED BY DEFINITIONS OF THE PARAMETERS USED, AND THEIR DEFAULT VALUES. IF NO DEFAULT VALUE IS SPECIFIED, THE DEFAULT IS BLANK FOR A NAME FIELD; ZERO FOR A NUMERIC FIELD.

INDENTATION IS USED IN THE KEYWORD FIELD AS A MATTER OF STYLE
ONLY, TO CONVEY "BELONGS TO" OR "SUBORDINATE TO" RELATIONSHIPS.

3.1 HARDWARE

THE HARDWARE TABLES DESCRIBE THE NUMBER, TYPES, AND CONFIGURATIONS OF THE HARDWARE ELEMENTS (CHANNELS, CONTROL UNITS, AND DEVICES) TO BE INCLUDED IN THE SYSTEM.

EACH CHANNEL IS LISTED AND, UNDER THAT CHANNEL, EACH OF THE CONTROL UNITS ATTACHED TO THE CHANNEL. SIMILARLY, UNDER EACH CONTROL UNIT ARE LISTED THE DEVICES ATTACHED TO THE CONTROL UNIT. A DEVICE IS CONSIDERED TO BE A SINGLE DRIVE; THAT IS, A 2314 FACILITY WOULD CONSIST OF EIGHT DEVICES ATTACHED TO ONE CONTROL UNIT.

CONTROL UNITS MAY BE SWITCHABLE BETWEEN CHANNELS. TO INDICATE SUCH AN OPTION, A CONTROL UNIT MAY BE LISTED UNDER MORE THAN ONE CHANNEL. HOWEVER, THE ATTACHED DEVICES MUST BE LISTED ONLY ONCE.

CHANNELS AND UNITS NEED NOT BE EXPLICITLY SPECIFIED. IF THEY ARE NOT, AN IMPLIED (NAMELESS) UNIT AND/OR CHANNEL WILL BE SUPPLIED BY THE SYSTEM.

```
*********************************************************************
        *HARDWARE
        NAME CHANNEL
        NAME   UNIT
        NAME     DEVICE TYPE,TRKP
                    .
                    .
                    .
               UNIT
                 .
                 .
                 .
             CHANNEL
                 .
                 .
                 .
             END
*********************************************************************
```

PARAMETER DEFINITIONS

PARM	DEFINITION	DEFAULT
NAME	NAME OF CHANNEL, UNIT, OR DEVICE	
TYPE	DEVICE TYPE. THERE ARE FOUR BUILT-IN TYPES: 2302,2311,2314,2321	

```
TRKP  INITIAL POSITION OF THE HEAD RELATIVE TO ZERO.
      THIS PROVIDES DIFFERENT RELATIVE ROTATIONAL
      POSITIONING ACROSS ACCESS MECHANISMS.
      0.<=TRKP<=1.
```

3.2 DEVICE TYPE

A DEVICE TYPE IS A COLLECTION OF PARAMETERS REPRESENTING THE PHYSICAL CHARACTERISTICS OF A KIND OF DEVICE; FOR EXAMPLE, 2311, 2321, 2314, AND 2302 (INCIDENTALLY THESE FOUR ARE BUILT INTO THE SYSTEM AND NEED NOT BE SUPPLIED BY THE USER). EACH DEVICE IN THE HARDWARE DESCRIPTION MUST ADOPT ITS CHARACTERISTICS FROM ONE OF THE DEVICE TYPES.

IT IS ASSUMED THAT THE CYLINDERS OF A DEVICE ARE NUMBERED 1 - N, AND CAN BE DIVIDED INTO ACCESS ZONES, EACH HAVING THE SAME NUMBER OF (CONTIGUOUS) CYLINDERS. FURTHERMORE, IT IS ASSUMED THAT THE ACCESS ZONES CAN BE SIMILARLY SUBDIVIDED INTO SUB-ACCESS ZONES. THIS ZONATION FORMS THE BASIS FOR DESCRIBING DELAYS DUE TO ACCESS ARM MOVEMENT BETWEEN CYLINDERS OF THE DEVICE. THESE "ACCESS TIMES" ARE A FUNCTION OF THE NUMBER AND TYPES OF ZONE BOUNDARIES PASSED OVER, AND, FOR EACH TYPE OF ZONE, CAN BE EXPRESSED AS A SINGLE SCALAR VALUE OR A TABLE OF VALUES. IF THE DEVICE HAS NO ZONATION (OTHER THAN CYLINDERS), ONLY ONE SCALAR OR ONE TABLE NEED BE SPECIFIED.

```
************************************************************************

        *DEVICE TYPE
        NAME DEVICE    PER,TRKC,NCYC,CAT,TB,DRAT,TPC,
                       DCV,KCV,VOC,
                       NCYA,CATA,TABA,NCYS,CATS,TABS
                .
                .
                .
             END
************************************************************************
```

PARAMETER DEFINITIONS

PARM	DEFINITION	DEFAULT
NAME	NAME OF DEVICE TYPE	
PER	ROTATIONAL PERIOD OF DEVICE IN MS.	
TRKC	TRACK CAPACITY - MAX SIZE (IN BYTES) RECORD THAT CAN BE STORED ON ONE TRACK	
NCYL	NO. CYLS. PER DEVICE	
CAT	BASIC CYLINDER ACCESS TIME IN MS.	TB USED
TB	BASIC CYLINDER ACCESS TIME TABLE	CAT USED
DRAT	DATA RATE IN BYTES/MS	

TPC	NO. TRACKS PER CYLINDER	
DOV	HARDWARE OVERHEAD FOR DATA - BYTES PER BLOCK	
KOV	HARDWARE OVERHEAD FOR KEY - BYTES PER BLOCK	
VOC	HARDWARE OVERHEAD VARIABLE - BYTES PER BLOCK	
NCYA	NO. CYLINDERS PER ACCESS ZONE	
CATA	CYL. ACCESS TIME BETWEEN ACCESS ZONES (MS)	TABA
TABA	TABLE OF CATA	CATA
NCYS	NO. CYLINDERS PER SUB-ACCESS ZONE	
CATS	CYL. ACCESS TIME BETWEEN SUB-ACCESS ZONES BUT WITHIN ACCESS ZONES	TABS
TABS	TABLE OF CATS	CATS

TB, TABA, TABS ARE TABLES IN WHICH

ARG= NO. OF CYLINDER, ACCESS ZONE, OR SUBACCESS ZONE BOUNDARIES, RESPECTIVELY, TO BE PASSED OVER BY THE ACCESS MECHANISM.

VAL= TIME IN MS. FOR THE ACCESS MECHANISM TO PERFORM THIS MANEUVER

USE OF A SCALAR (CAT,CATA,CATS) INSTEAD OF THE TABLE IMPLIES THAT THE TIME TAKEN IS INDEPENDENT OF THE NUMBER OF BOUNDARIES CROSSED.

THE NUMBER OF BLOCKS THAT CAN BE ACCOMMODATED BY A TRACK IS COMPUTED BY THE FOLLOWING FORMULA:

$$1 + (TRKC - KOV - KL - BLOCKSIZE)/(BLOCKSIZE + DOV + KOV + VOC * (BLOCKSIZE + KL))$$

WHERE KL=0 IMPLIES KOV=0 (KL = KEY LENGTH).

3.3 SEGMENTS

THE SEGMENT SPECIFICATION IS THE MEANS BY WHICH THE LOGICAL ORGANIZATION OF THE DATABASE IS DESCRIBED, AND BY WHICH THE ASSIGNMENT OF SEGMENTS TO DATASETS IS MADE. IT IS ALSO THE BASIS FOR PHRASING QUALIFICATION SPECIFICATIONS.

A SEGMENT IS A COLLECTION OF FIELDS DESCRIBING AN ENTITY. FOR EXAMPLE, NAME, AGE, AND SEX MAY BE USED TO DESCRIBE A PERSON. DIFFERENT KINDS OF ENTITIES (FOR EXAMPLE, PEOPLE, ORGANIZ- ATIONS, AND BOOKS) CAN BE REPRESENTED IN THE SAME SYSTEM, EACH HAVING ITS OWN SEGMENT TYPE AND COLLECTION OF FIELDS.

A HIERARCHICAL OR TREE DATA STRUCTURE IS ASSUMED: THAT IS, EACH SEGMENT TYPE MAY HAVE ONE OR MORE "INFERIOR" SEGMENT TYPES, EACH OF WHICH OCCURS A GIVEN NUMBER OF TIMES FOR EACH OCCUR- RENCE OF ITS SUPERIOR SEGMENT.

EACH SEGMENT TYPE IS ASSOCIATED WITH A GIVEN DATASET. THIS ASSOCIATION MEANS THAT ALL OCCURRENCES OF THAT SEGMENT WILL BE ASSIGNED TO (STORED IN) THE SPECIFIED DATASET.

THE "DATASET MASTER SEGMENT" OF A SEGMENT IS THE HIGHEST LEVEL SEGMENT SUPERIOR TO IT AND ON THE SAME DATASET. ALL SEGMENT TYPES ASSIGNED TO A GIVEN DATASET MUST HAVE THE SAME DATASET MASTER TYPE.

A "DATASET" RECORD CONSISTS OF A DATASET MASTER SEGMENT TOGETHER WITH ALL SEGMENTS INFERIOR TO IT THAT HAVE ALSO BEEN ASSIGNED TO THAT DATASET.

A FIELD MAY BE A SORT FIELD; THAT IS, FIELD VALUES FOR A SORT FIELD WILL OCCUR IN THE DATASET IN THE ORDER IN WHICH THEY ARE PRESENTED IN THE DISTRIBUTION OF THE FIELD. THE VALUES OF NON-SORT FIELDS ARE ASSUMED TO BE UNIFORMLY DISTRIBUTED THROUGHOUT THE OCCURRENCES OF THE FIELD. A SORT FIELD MAY OCCUR ONLY IN A DATASET MASTER SEG- MENT, AND EACH DATASET MAY HAVE AT MOST ONE SORT FIELD.

```
*****************************************************************************

     *SEGMENTS
     NAME SEGMENT   SIZE,SUP,DS,NPSS
     NAME   FIELD   SIZE,DIST,TYPE,SIDS
              .
              .
              .
          SEGMENT
              .
              .
              .
          END

*****************************************************************************
```

```
              PARAMETER DEFINITIONS

PARM  DEFINITION                                              DEFAULT
----------------------------------------------------------------------

NAME  NAME OF SEGMENT OR FIELD. IT IS NOT NECESSARY TO LIST
      THE FIELDS OF A SEGMENT IF THEY ARE NOT GERMANE TO THE
      SPECIFICATION.

SIZE  FIELD SIZE OR INITIAL SEGMENT SIZE.
      AFTER INPUT, FOR SEGMENT THIS BECOMES:
      INITIAL SEGMENT SIZE + SUM OF FIELD SIZES IN SEGMENT

SUP   SUPERIOR SEGMENT NAME

DS    DATASET TO WHICH THE SEGMENT IS ASSIGNED

NPSS  NUMBER OF THESE SEGMENTS PER SUPERIOR SEGMENT.
      FOR A SEGMENT WITH NO SUPERIOR SEGMENT, THIS
      WILL BE THE TOTAL NUMBER OF THESE SEGMENTS (AND THE
      NUMBER OF RECORDS ON THE DATASET).

DIST  NAME OF THE DISTRIBUTION (IN THE DISTRIBUTION TABLE),
      OF THE VALUES OF THIS FIELD.

TYPE  FIELD TYPE
      S = SORT FIELD
      SI=  SECONDARY INDEX KEY FIELD  (NOT IMPLEMENTED)

SIDS  NAME OF SECONDARY INDEX DATASET FOR THIS FIELD
      (NOT IMPLEMENTED)
```

3.4 DATASETS

A DATASET IS A LOGICAL COLLECTION OF RECORDS NUMBERED 1-N, AND IS THE PRIMARY VEHICLE FOR REFERRING TO AND ACCESSING DATA. EACH DATASET CONSISTS OF ONE OR MORE FILES. THE SALIENT FEATURE OF A FILE IS THAT IT HAS THE SAME PHYSICAL RECORD FORMAT THROUGHOUT; WHEREAS, FOR DIFFERENT FILES BELONGING TO A DATASET, THIS MAY NOT BE TRUE.

FOR EXAMPLE, AN INDEXED DATASET MAY CONSIST OF A PRIME DATA FILE, AN INDEX FILE, AND AN OVERFLOW FILE, ALL HAVING DIFFERENT RECORD LENGTHS AND BLOCKING FACTORS.

A STRICTLY SEQUENTIAL DATASET CONSISTS OF ONLY ONE FILE. SUCH DATASETS ARE CALLED "ONE-FILE" DATASETS.

A FILE IS SUBDIVIDED INTO "EXTENTS". THIS ALLOWS A FILE TO BE SCATTERED OVER SEVERAL DEVICES, OR TO OCCUPY NON-CONTIGUOUS AREAS ON THE SAME DEVICE. AN EXTENT IS CHARACTERIZED BY ITS DEVICE, FIRST CYLINDER, AND NUMBER OF CYLINDERS.

MOST OF THE PARAMETERS DESCRIBED HAVE DEFAULT VALUES. IN FACT, FILE AND EXTENT DESCRIPTIONS MAY BE OMITTED ENTIRELY WHERE THE DEFAULTS SUIT THE USER.

IF THE EXTENTS SPECIFIED BY THE MODELER WILL NOT CONTAIN THE FILE, ADDITIONAL EXTENTS WILL BE PROVIDED FROM UNOCCUPIED DEVICES OF THE TYPE ON WHICH THE FILE IS TO RESIDE. SUCH DEFAULT-ASSIGNED DEVICES WILL BECOME UNAVAILABLE FOR SUBSEQUENT DEFAULT ALLOCATION (EVEN THOUGH THE OCCUPYING FILE DOES NOT USE THE WHOLE DEVICE).

A FILE MAY "SHARE" EXTENTS WITH ANOTHER FILE; THAT IS, IT MAY OCCUPY THE SAME CYLINDERS AS ANOTHER FILE (BUT DIFFERENT TRACKS).

FOR A DEFINITION OF THE ACCESS METHOD DEFINED PARAMETERS FOR EACH ACCESS METHOD, SEE SECTION 7.

```
**********************************************************************

    *DATASETS
    NAME DATASET     TYPE,NREC,RSIZ,DEVT
         PARAM       ACCESS METHOD DEFINED PARAMETERS
    NAME   FILE      TYPE,DEVT,RPB,TPC,ALLT,ALL,BTYP,NBUF,
                     WV,CH,EXT,RPC,KL
             EXTENT  DEV,CYL,NCYL
                .
                .
                .
           FILE
             .
             .
             .
```

```
      DATASET
          .
          .
          .
      END
******************************************************************************

                      PARAMETER DEFINITIONS

        PARM  DEFINITION                                         DEFAULT
        ----------------------------------------------------------------

        NAME  NAME OF DATASET OR FILE

        TYPE  FOR DATASET, ACCESS METHOD TYPE. FOR A DESCRIPTION       S
              OF THE ACCESS METHODS, SEE SECTION 7.

              FOR FILE, IDENTIFICATION OF FILE TYPE. ALLOWABLE FILE
              TYPES DEPEND ON THE TYPE OF DATASET TO WHICH THE FILE
              BELONGS. THIS PARAMETER IS IGNORED IF THE DATASET IS A
              ONE-FILE DATASET.

        NREC  NO. OF RECORDS ON DATASET. IF SEGMENTS ARE
              ASSIGNED TO THIS DATASET, "NREC" IS TAKEN TO BE
              THE NUMBER OF DATASET MASTER SEGMENTS.  IF
              NO SEGMENTS ARE ASSIGNED TO THE DATASET, AND "NREC" IS
              NOT SPECIFIED, IT WILL BE COMPUTED TO BE THE NUMBER OF
              RECORDS THAT WILL BE ACCOMMODATED BY THE SPACE ALLOCATED
              TO THE DATASET, EITHER THROUGH THE "ALLT" AND "ALL"
              PARAMETERS OF THE ASSOCIATED FILE, OR BY THE EXTENTS
              PROVIDED BY THE USER (ONE-FILE DATASETS ONLY).

        RSIZ  RECORD SIZE. THIS SIZE IS ADDED TO THE
              CONTRIBUTION IN RECORD SIZE DUE TO SIZES OF
              SEGMENTS (IF ANY) ASSIGNED TO THE DATASET.

        DEVT  DATASET DEFAULT DEVICE TYPE. A SPECIFICATION HERE WILL
              CAUSE ALL FILES ASSOCIATED WITH THE DATASET TO BE
              ASSIGNED TO DEVICES OF THIS TYPE, IF THEY HAVE NOT BEEN
              OTHERWISE ASSIGNED.

              DEVICE TYPE TO WHICH THE FILE IS TO BE ASSIGNED.
              THIS PARAMETER MAY BE OMITTED IF ASSIGNMENT TO
              SPECIFIC DEVICES IS MADE BY USE OF EXTENTS, OR
              THIS FILE SHARES EXTENTS WITH ANOTHER FILE,
              OR ASSIGNMENT TO THE "DEVT" SPECIFIED BY THE
              DATASET IS DESIRED. A FILE MAY RESIDE ON ONLY ONE
              TYPE OF DEVICE.

        RPB   NO. RECORDS PER BLOCK                                    1

        TPC   NO. TRACKS PER ALLOCATED CYLINDER TO BE ASSIGNED
              TO THIS FILE.                                   DEVICE TPC
```

ALLT	ALLOCATION UNIT: REC FOR ALLOCATION IN RECORDS TRK FOR ALLOCATION IN TRACKS CYL FOR ALLOCATION IN CYLINDERS	REC
ALL	NO. OF THE ABOVE UNITS TO BE ALLOCATED. DEFAULT: ENOUGH TO ACCOMMODATE "NREC" RECORDS,	
BTYP	BUFFERING TYPE FOR SEQUENTIAL ACCESS. "M" = "MOVE", "L" = "LOCATE", AS DESCRIBED BY OS/360 DATA MANAGEMENT.	
NBUF	NO. OF BUFFERS TO BE ASSIGNED WHEN THIS FILE IS OPENED. BUFFER SIZE IS DICTATED BY BLOCKSIZE.	
WV	"WV" IF WRITE VERIFICATION IS TO BE PERFORMED FOR THIS FILE (NOT CURRENTLY IMPLEMENTED)	
CH	"CH" IF COMMAND CHAINING IS TO BE USED (NOT CURRENTLY IMPLEMENTED)	
EXT	NAME OF FILE WITH WHICH THIS FILE IS TO SHARE EXTENTS	
RPC	NO. OF RECORDS OF THIS DATASET TO BE ASSIGNED PER ALLOCATED CYLINDER. DEFAULT: NO. OF RECORDS THAT CAN BE ACCOMMODATED BY "TPC" TRACKS.	
KL	KEY LENGTH	
DEV	NAME OF THE DEVICE THE EXTENT RESIDES ON	
CYL	CYLINDER OF THE DEVICE ON WHICH THE EXTENT BEGINS	
NCYL	NUMBER OF CONTIGUOUS CYLINDERS IN THE EXTENT. DEFAULT: NO. OF CYLINDERS ON OCCUPIED DEVICE.	

3.5 QUALIFICATIONS

THE QUALIFICATION SPECIFICATIONS ARE ROUGHLY EQUIVALENT TO QUERIES ON THE DATABASE. EACH QUALIFICATION DESCRIBES A CRITERION WHICH A SEGMENT MUST MEET IN ORDER TO QUALIFY, AND RESULTS IN A LIST OF QUALIFIED RECORDS ON THE ASSOCIATED DATASET. THESE LISTS ARE MADE ACCESSIBLE TO THE PROCEDURE THROUGH THE "SQ" AND "RQ" TYPE LISTS, OR THE "Q" MODIFIER IN A PROCEDURE ACCESS OPERATION (SEE PROCEDURE SPECIFICATION).

THERE ARE THREE TYPES OF QUALIFICATION SPECIFICATION: "FIELD", "BOOLEAN", AND "SEGMENT".

IN THE FOLLOWING, "FLD" REPRESENTS A FIELD NAME, "Q1" AND "Q2" REPRESENT QUALIFICATION LABELS, AND "SEG" REPRESENTS A SEGMENT NAME.

EACH QUALIFICATION IS A QUALIFICATION ON A UNIQUE SEGMENT TYPE, AS FOLLOWS:

A FIELD QUALIFICATION IS ON THE SEGMENT CONTAINING "FLD".

A BOOLEAN QUALIFICATION IS ON THE SEGMENT QUALIFIED BY "Q1" AND "Q2", WHICH MUST QUALIFY THE SAME SEGMENT.

A SEGMENT QUALIFICATION IS ON THE SEGMENT NAMED BY "SEG".

IN TURN, A QUALIFYING SEGMENT QUALIFIES THE RECORD THAT IT BELONGS TO IN THE DATASET CONTAINING THAT RECORD. WHEN THE QUALIFICATION TABLES ARE PRINTED BY THE PROCEDURE, THREE ADDITIONAL PARAMETERS, "LRQ", "HRQ", AND "NRQ" ARE ALSO PRINTED FOR EACH QUALIFICATION. THESE REPRESENT, RESPECTIVELY, THE "LOW RECORD", "HIGH RECORD", AND "NUMBER OF RECORDS" QUALIFIED ON THE APPROPRIATE DATASET.

A QUALIFICATION IS INTERPRETED AS FOLLOWS BY TYPE (SEE THE PROTOTYPE QUALIFICATIONS BELOW):

FIELD	- A SEGMENT CONTAINING FIELD "FLD" QUALIFIES IF "FLD" BEARS RELATION "REL" TO VALUE "VAL".
BOOLEAN	- A SEGMENT QUALIFIES IF IT ALSO QUALIFIES BY "Q1" "REL1" "Q2".
SEGMENT	- SEGMENT "SEG" QUALIFIES IF IT HAS "REL2" "N" SEGMENTS THAT QUALIFY BY "Q3".

A FIELD QUALIFICATION ON A SORT FIELD RESULTS IN QUALIFICATION OF A SUBSET OF THE RANGE OF RECORDS IN A DATASET. FURTHER USE OF SUCH A QUALIFICATION BY A SEGMENT QUALIFICATION (WHERE THE SEGMENT BEING QUALIFIED IS IN THE SAME DATASET AS THE GIVEN FIELD) WILL RESULT IN DERIVATIVE QUALIFYING SUBSETS OF THE SAME TYPE. SUCH A QUALIFICATION IS CALLED A "SORT QUALIFIC-ATION", AND AN "OR" SEGMENT QUALIFICATION IS NOT PERMITTED IF EITHER "Q1" OR "Q2" IS OF THIS TYPE.

```
*************************************************************************

     *QUALIFICATIONS
     NAME QUAL       FLD,REL,VAL          (FIELD QUALIFICATION)
     NAME QUAL       Q1,REL1,Q2           (BOOLEAN QUALIFICATION)
     NAME QUAL       SEG,HAS,Q3,REL2,N    (SEGMENT QUALIFICATION)
              .
              .
              .
             END

*************************************************************************
```

PARAMETER DEFINITIONS

```
     PARM  DEFINITION                                           DEFAULT
     -----------------------------------------------------------------

     NAME  NAME OF QUALIFICATION

     FLD   FIELD NAME

     REL   RELATION:  "EQ" FOR "EQUALS"
                      "LE" FOR "LESS THAN OR EQUAL TO"
                      "GT" FOR "GREATER THAN"

     VAL   A VALUE OF THAT FIELD

     Q1,Q2 NAMES OF QUALIFICATIONS. Q1,Q2 MUST BE QUALIFIC-
           ATIONS ON THE SAME SEGMENT

     REL1  "AND" OR "OR"

     SEG   NAME OF A SEGMENT

     HAS   A LITERAL "HAS"

     Q3    NAME OF A QUALIFICATION. THE SEGMENT QUALIFIED BY
           Q3 MUST BE LINEALLY RELATED TO "SEG", THAT IS,
           ONE MUST BE A DIRECT DESCENDANT OF THE OTHER IN
           THE SEGMENT HIERARCHY.

     REL2  EQ,LE,GT - SAME INTERPRETATION AS FOR REL             GT
           ALL      - ALL SEGMENTS RELATED TO "SEG" MUST
                      QUALIFY BY "Q3"

     N     A NON-NEGATIVE INTEGER
```

3.6 PROCEDURE

THE PROCEDURE IS THE MEANS BY WHICH THE USER INSTRUCTS THE SYSTEM WHAT IS TO BE DONE WITH THE CONFIGURATIONS OF HARDWARE, SOFTWARE, DATASETS AND QUERIES DESCRIBED. IT ALSO PROVIDES CERTAIN MODEL CONTROL AND DEBUGGING FACILITIES.

INTERPRETATION AND DEFAULT VALUES OF PARAMETERS ARE DEPENDENT ON THE PROCEDURE OPERATION TYPE.

```
*************************************************************************

        *PROCEDURE
        LBL  OP      M OBJ,LIST,SGO,FGO,TIME
              .
              .
              .
             END

*************************************************************************
```

PARAMETER DEFINITIONS

PARM	DEFINITION	DEFAULT
LBL	STATEMENT LABEL	
OP	OPERATION TO BE PERFORMED (COLS. 6-14) MUST BE SEPARATED FROM A MODIFIER, IF PRESENT, BY ONE OR MORE BLANKS	
M	(MOD) MODIFIER, WHICH ALTERS THE MEANING OF THE OTHER PARAMETERS FOR SOME OPERATIONS (COL 15). JUST LEAVE BLANK TO OMIT; DO NOT CODE A COMMA.	
OBJ	OBJECT OF OPERATION	
LIST	LIST TO BE USED BY THE OPERATION. IT MAY APPEAR AS A LITERAL LIST, IN THE FORM (X1,X2,...,XN).	
SGO	LABEL TO GO TO IF OPERATION "SUCCEEDS"	NEXT STMT
FGO	LABEL TO GO TO IF OPERATION "FAILS"	NEXT STMT
TIME	CPU TIME IN MS. TO BE ASSOCIATED WITH THIS OPERATION. IT IS APPLIED WHEN THE OPERATION IS COMPLETE.	

OPERATIONS WHICH MAY BE USED IN "OP" FIELD:

PRINT   PRINT ON LOGICAL FILE "OBJ" (DEFAULT= STANDARD

OUTPUT) THE TABLES NAMED IN "LIST". ELEMENTS OF THE LIST CAN BE:

ALL	PRINT ALL TABLES
DI	DISTRIBUTIONS
TB	TABLES
DP	DEVICE TYPES (OR PROTOTYPES)
HD	HARDWARE CONFIGURATION AND PARAMETERS
SG	SEGMENT (LOGICAL STRUCTURE OF DATA)
PR	PROCEDURE
LI	LISTS
DS	DATASETS (PHYSICAL STRUCTURE OF DATA)
QU	QUALIFICATIONS
TI	TIMERS
IO	I/O STATUS

FOR SOME TABLES, SOME INTERNAL PARAMETERS ARE PRINTED OUT TO PROVIDE ADDITIONAL INFORMATION FOR THE MODELER. INTERNAL PARAMETERS ARE DEFINED IN SECTION 10 UNDER THE APPROPRIATE TABLE ENTRY.

DUMP — DUMP (PRINT EXTERNAL AND INTERNAL PARAMETERS) ON LOGICAL FILE "OBJ" (DEFAULT= STANDARD OUTPUT) TABLES NAMED IN "LIST", WHICH MAY INCLUDE ANY OF THE ABOVE, PLUS:

DV	DEVICES
CU	CONTROL UNITS
CH	CHANNELS
LD	LOGICAL DATA (SEGMENTS AND FIELDS)
FD	FIELDS
PD	PHYSICAL DATA (DATASETS, FILES, EXTENTS)
FL	FILES
EX	EXTENTS
UT	UTILITIES (DISTRIBUTIONS, TABLES, LISTS)
BU	BUFFERS
Q	I/O QUEUES

TRACE — TRACE SUBROUTINES NAMED IN "LIST". A SUBROUTINE IS IDENTIFIED BY A FOUR CHARACTER STRING CONSISTING OF THE FIRST TWO AND THE LAST TWO CHARACTERS OF THE SUBROUTINE NAME. THE TRACE WILL APPEAR ON THE STANDARD OUTPUT. USE OF "EVNT" AS A SUBROUTINE NAME WILL CAUSE TRACING OF ALL I/O EVENTS. USE OF "ALL" WILL CAUSE ALL SUBROUTINES (AND I/O EVENTS) TO BE TRACED. SEE SECTION 9.3 FOR A LIST OF ALL SUBROUTINES IN THE SYSTEM, AND THEIR FUNCTIONS. ASSEMBLER LANGUAGE ROUTINES ARE NOT TRACEABLE.

NOTRACE — SUSPEND TRACING OF THE SUBROUTINES NAMED IN "LIST"

STRACE — AFTER EVERY 24 PROCEDURE STATEMENTS EXECUTED, PRINT THE 24 STATEMENT NUMBERS ON LOGICAL FILE "OBJ" (DEFAULT= STANDARD OUTPUT)

ERROR	IF AN ERROR OCCURS DURING EXECUTION, TRANSFER TO THE LABEL NAMED BY "OBJ". IF AN ERROR OCCURS AND NO "ERROR" STATEMENT HAS BEEN EXECUTED, THE LAST "ERROR" STATEMENT IN THE PROCEDURE WILL PRESCRIBE THE ACTION TO BE TAKEN.
	IF "OBJ" IS BLANK, THE STATEMENT FOLLOWING THE ERROR STATEMENT IS ASSUMED.

RESTORE	RESTORE THE SYSTEM TO TIME=0
TIME	SET TIMER "OBJ" TO ZERO
PTIME	PRINT TIMER "OBJ" (MS. OF ELAPSED SIMULATED TIME SINCE IT WAS LAST SET). ALL TIMERS ARE AUTOMATICALLY SET TO ZERO AT PROCEDURE INITIATION, AND BY "RESTORE".
END	END OF PROCEDURE. THIS DELIMITS PROCEDURE STATEMENTS, AND WHEN TRANSFERRED TO, WILL END PROCEDURE EXECUTION.
(BLANK)	NO OPERATION, BUT HONOR "SGO" AND "TIME" PARAMETERS
INIT	RE-INITIALIZE LIST "OBJ", AND/OR EACH LIST IN "LIST" TO ITS STATE AT PROCEDURE INITIATION.
SYNC	"SYNCHRONIZE" THE OPERATIONS NAMED AT THE LABELS IN THE LIST. THAT IS, A TRANSFER TO ONE OF THE SPECIFIED LABELS CAN RESULT IN A TRANSFER TO ANY ONE OF THE LABELS. THE PROBABILITY THAT A GIVEN LABEL WILL BE TRANSFERRED TO IS PROPORTIONAL TO THE CURRENT LENGTH OF THE LIST NAMED BY THE LIST ARGUMENT AT THAT PRO-CEDURE LABEL. "OBJ" SPECIFIES THE LABEL TO BE BRANCHED TO WHEN ALL THE LISTS ARE EMPTY. THIS OPERATION PROVIDES PARALLEL OPERATIONS ON DATASETS WHEN THERE IS NO FIXED PATTERN OF ACCESSES, FOR EXAMPLE, IN MERGE OPERATIONS. "SYNC" IS A PROCEDURE CONTROL OPERATION. ITS EXECUTION HAS NO IMMEDIATE AFFECT EXCEPT TO INDICATE TO THE SYSTEM THAT THE SPECIFIED STATEMENTS ARE TO OPERATE IN "SYNC" MODE.
READS WRITS	READ OR WRITE (USING SEQUENTIAL PROCESSING) A RECORD OF THE DATASET NAMED BY "OBJ". THE RECORD TO BE ACCESSED IS DEFINED BY THE NEXT NUMBER ON THE SPEC-IFIED LIST. SUCCESSIVE ACCESSES WILL BE MADE TO THE DATASET (STARTING AT RECORD 1 IF THE DATASET IS NOT OPEN) UNTIL THE REQUIRED RECORD IS ACCESSED, OR AN END-OF-DATA INDICATION IS RETURNED.

THE RECORD NUMBER TO BE ACCESSED IS REMOVED FROM THE LIST.

THE OPERATION SUCCEEDS IF THE REQUIRED RECORD CAN BE ACCESSED, AND TRANSFER IS MADE TO THE "SGO" LABEL.

THE OPERATION FAILS IF THE LIST IS EMPTY, OR END-OF-DATA IS REACHED ON THE DATASET.

DEFAULT FOR "LIST" IS A SEQUENTIAL LIST 1,2,...,NREC WHERE NREC IS THE NUMBER OF RECORDS IN THE DATASET

IF MOD= "F", THE DATASET IS TAKEN TO BE THE DATASET ON WHICH THE FIELD NAMED BY "OBJ" RESIDES.

IF MOD= "Q", THE DATASET IS TAKEN TO BE THE DATASET ON WHICH THE SEGMENT QUALIFIED BY THE QUALIFICATION NAMED IN "OBJ" RESIDES. THE LIST WILL BE INFERRED TO BE A RANDOM/SEQUENTIAL ARRANGEMENT OF THE RECORDS OF THE DATASET QUALIFIED BY THE NAMED QUALIFICATION, AND THE LIST PARAMETER SHOULD NOT APPEAR EXPLICITLY.

THE "F" AND "Q" MODIFIERS MAY BE USED WITH ANY OPERATION WHOSE OBJECT IS A DATASET.

READR
WRITR

READ OR WRITE (USING RANDOM PROCESSING) A RECORD OF THE DATASET NAMED BY "OBJ". THE RECORD TO BE ACCESSED IS DEFINED BY THE NEXT NUMBER ON THE SPECIFIED LIST.

THE RECORD NUMBER TO BE ACCESSED IS REMOVED FROM THE LIST, AND THE OPERATION FAILS WHEN THE LIST IS EMPTY.

DEFAULT FOR "LIST" IS A RANDOM LIST OF NREC INTEGERS ON THE INTERVAL (1,NREC).

MOD HAS THE SAME INTERPRETATION AS FOR READS, EXCEPT THAT FOR MOD="Q", THE INFERRED LIST WILL BE A STRICTLY RANDOM LIST INSTEAD OF A RANDOM SEQUENTIAL LIST.

READD
WRITD

READ OR WRITE (USING DIRECT PROCESSING) A RECORD OF THE DATASET NAMED BY "OBJ". THE PARAMETERS ARE INTERPRETED AS THEY ARE FOR READR AND WRITR.

WAIT

SUSPEND PROCESSING UNTIL COMPLETION OF THE FIRST DIRECT I/O REQUEST ON DATASET "OBJ" FOR WHICH A "WAIT" HAS NOT BEEN ISSUED.

UPDATE

UPDATE THE LAST RECORD READ FROM DATASET "OBJ"

OPEN

OPEN DATASET "OBJ" WITH THE PARAMETERS GIVEN IN THE LIST. THESE PARAMETERS ARE (STATUS,NBUF,BTYP,CH,WV), WHERE:

```
                STATUS= R   FOR READ SEQUENTIAL
                        W   FOR WRITE SEQUENTIAL
                        X   FOR RANDOM PROCESSING

                THE REMAINING PARAMETERS ARE AS DEFINED IN THE
                DATASET INPUT PARAMETER DESCRIPTION.
      CLOSE     CLOSE DATASET "OBJ"
```

3.7 LISTS

THE LIST FACILITY PROVIDES A MEANS OF DEFINING LISTS FOR USE BY THE PROCEDURE SPECIFICATION. THESE PRIMARILY WILL BE LISTS OF RECORD NUMBERS TO BE ACCESSED FROM DATASETS IN THE SYSTEM. THE USER CAN SUPPLY LISTS WHOSE CONTENTS ARE EXPLICITLY DESCRIBED, OR HE MAY SPECIFY THAT A LIST IS TO BE DERIVED FROM A QUALIFICATION SPECIFICATION.

```
***********************************************************************

*LISTS
NAME LIST      TYPE,MQ,SIZE,LC,HS,DIST
               VAL1,VAL2,....,VAL20
               VAL21,VAL22,....,VAL40
                      .
                      .
                      .
               ... ,VALN
     LIST
      .
      .
      .
     END
***********************************************************************
```

PARAMETER DEFINITIONS

```
PARM  DEFINITION                                              DEFAULT
---------------------------------------------------------------------

NAME  LIST NAME

TYPE  LIST TYPE:
      LL  =  LITERAL LIST. LIST VALUES (VAL1,....,VALN) ARE
             TO BE LISTED ON SUCCEEDING CARDS.
      SL  =  SEQUENTIAL
      RL  =  RANDOM
      RS  =  RANDOM/SEQUENTIAL  (SORTED RANDOM)
      SQ  =  RANDOM/SEQUENTIAL BASED ON QUALIFICATION
      RQ  =  RANDOM BASED ON QUALIFICATION

MQ    MODE OF LIST, OR QUALIFICATION NAME FOR SQ, RQ TYPE
      LISTS. MODE CAN BE:

      I     INTEGER
      R     REAL
      A     ALPHAMERIC, ELEMENTS AT MOST FOUR CHARACTERS EACH

      FOR SQ, RQ TYPE, MQ CAN BE ANY QUALIFICATION NAME.

SIZE  SIZE OF LIST
```

LO	FOR SL, FIRST ELEMENT OF LIST FOR RL, RS, LOWER BOUND FOR ELEMENTS OF LIST
HS	FOR SL, SKIP FACTOR FOR RL, RS, UPPER BOUND FOR ELEMENTS OF LIST
DIST	FOR RL, DISTRIBUTION FROM WHICH ELEMENTS ARE TO BE TAKEN. DEFAULT = UNIFORM DISTRIBUTION ON (LO,HS)
VAL1	VALUES OF LITERAL LIST, 20 PER CARD, EXCEPT POSSIBLY THE LAST
.	
.	
.	
VALN	

3.8 TABLES

THE TABLE FACILITY PROVIDES A TABLE-LOOK-UP CAPABILITY. TABLES CAN HAVE ARGUMENTS AND VALUES WHICH ARE INTEGER, REAL, OR ALPHAMERIC (AT MOST FOUR CHARACTERS), AND MAY REPRESENT EITHER STEP-FUNCTIONS, OR FUNCTIONS REQUIRING INTERPOLATION WHEN A VALUE IS NEEDED CORRESPONDING TO AN ARGUMENT NOT EXPLICITLY LISTED.

IN THE CASE OF A STEP-FUNCTION AN (ARG,VAL) PAIR MEANS THAT ANY ARGUMENT GREATER THAN OR EQUAL TO ARG AND LESS THAN THE NEXT ARG IN THE TABLE HAS VALUE "VAL".

AN INTERPOLATION TABLE MAY NOT HAVE ALPHAMERIC ARG'S OR VAL'S.

```
*****************************************************************************

        *TABLES
        NAME TABLE      TYPE,ARGT,VALT
                        ARG,VAL   (ONE PAIR PER CARD)
                          .
                          .
                          .
            TABLE
              .
              .
              .
            END
*****************************************************************************
```

PARAMETER DEFINITIONS

PARM	DEFINITION	DEFAULT
NAME	NAME OF TABLE	

TYPE	S= STEP FUNCTION
	I= INTERPOLATE (LINEAR)	

ARGT	ARGUMENT TYPE:
	I= INTEGER
	R= REAL
	A= ALPHAMERIC	

VALT	VALUE TYPE (AS FOR ARGT)	
ARG	ARGUMENT	
VAL	VALUE ASSOCIATED WITH THE ARGUMENT	

3.9 DISTRIBUTIONS.

THE DISTRIBUTION FACILITY ALLOWS THE USER TO SPECIFY DISTRIBUTIONS, CHIEFLY FOR DESCRIBING THE DISTRIBUTION OF THE VALUES OF A FIELD, AND DISTRIBUTIONS OF RECORD NUMBERS TO BE ACCESSED FROM DATASETS.

```
*************************************************************************

       *DISTRIBUTIONS
       NAME DIST       TYPE,MODE
                       ARG,VAL    (ONE PAIR PER CARD)
                         .
                         .
                         .
            DIST
             .
             .
             .
            END
*************************************************************************
```

PARAMETER DEFINITIONS

PARM	DEFINITION	DEFAULT
NAME	NAME OF DISTRIBUTION	

TYPE	C= CONTINUOUS FOR REAL DISTRIBUTION
	= INTERPOLATE FOR INTEGER DISTRIBUTIONS
	
	D= DISCRETE FOR REAL DISTRIBUTIONS
	= NO-INTERPOLATE FOR INTEGER DISTRIBUTIONS	

MODE	RANDOM VARIABLE TYPE:
	I= INTEGER
	R= REAL
	A= ALPHAMERIC	

ARG	DISTRIBUTION ARGUMENT	
VAL	DISTRIBUTION VALUE	

RULES:

1. IF TYPE=C, ARG'S MUST BE STRICTLY INCREASING
2. IF MODE=A, TYPE MUST BE D.
3. IF LAST VAL= 1., DISTRIBUTION IS ASSUMED TO BE IN CUMULATIVE FORM. AFTER INPUT, ALL DISTRIBUTIONS WILL BE STORED IN CUMULATIVE FORM.
4. IF TYPE=C, FIRST VAL MUST BE O.

3.10 TABLE SUMMARY

FOLLOWING IS A SUMMARY OF SPECIFIABLE INPUT TABLES AND THEIR INPUT PARAMETERS.

```
*********************************************************************

      *HARDWARE
      NAME CHANNEL
      NAME   UNIT
      NAME     DEVICE TYPF,TRKP
                    .
                    .
                    .
             UNIT
               .
               .
               .
           CHANNEL
               .
               .
               .
           END

*********************************************************************

      *DEVICE TYPF
      NAME DEVICE      PER,TRKC,NCYC,CAT,TB,DRAT,TPC,
                       DDV,KDV,VCC,
                       NCYA,CATA,TABA,NCYS,CATS,TABS
               .
               .
               .
           END

*********************************************************************

      *SEGMENTS
      NAME SEGMENT     SIZE,SUP,CS,NPSS
      NAME   FIELD     SIZE,DIST,TYPE,SIDS
               .
               .
               .
           SEGMENT
               .
               .
               .
           END

*********************************************************************
```

```
*DATASETS
NAME CATASET     TYPE,NREC,RSIZ,DEVT
     PARAM       ACCESS METHOD DEFINED PARAMETERS
NAME   FILE      TYPE,DEVT,RPB,TPC,ALLT,ALL,BTYP,NBUF,
                 WV,CH,EXT,RPC,KL
         EXTENT DEV,CYL,NCYL
           .
           .
           .
       FILE
        .
        .
        .
     DATASET
        .
        .
        .
     END

**********************************************************************

*QUALIFICATIONS
NAME QUAL       FLD,REL,VAL          (FIELD QUALIFICATION)
NAME QUAL       Q1,REL1,Q2           (BOOLEAN QUALIFICATION)
NAME QUAL       SEG,HAS,Q3,REL2,N    (SEGMENT QUALIFICATION)
        .
        .
        .
     END

**********************************************************************

*PROCEDURE
LBL  OP      M OBJ,LIST,SGO.FGO,TIME
     .
     .
     .
     END

**********************************************************************

*LISTS
NAME LIST       TYPE,MQ,SIZE,LC,HS,DIST
                VAL1,VAL2,...,VAL20
                VAL21,VAL22,...,VAL40
                       .
                       .
                       .
                ... ,VALN
     LIST
        .
        .
```

```
          .
          END

**************************************************************************

      *TABLES
      NAME TABLE        TYPE,ARGT,VALT
                        ARG,VAL    (ONE PAIR PER CARD)
                          .
                          .
                          .
          TABLE
            .
            .
            .
          END

**************************************************************************

      *DISTRIBUTIONS
      NAME DIST         TYPE,MODE
                        ARG,VAL    (ONE PAIR PER CARD)
                          .
                          .
                          .
          DIST
            .
            .
            .
          END
```

4.0 EXAMPLE MODEL SPECIFICATIONS

THE FOLLOWING PAGES CONTAIN SOME SAMPLE PROGRAMS, OUTPUTS, AND EXPLANATORY MATERIAL. EACH EXAMPLE CONSISTS OF A DESCRIPTION OF THE SYSTEM AND PROCESS TO BE MODELLED, NOTES ON THE SPEC-IFICATION AND OUTPUT OF THE MODEL, AND THE ACTUAL MODEL SPECIFICATIONS AND RESULTS. THE SAMPLE PROGRAMS WERE RUN ON THE IBM RESEARCH 360/91 COMPUTER AT SAN JOSE, CALIFORNIA. THE SIMULATED TIMINGS ARE NOT INTENDED TO REFLECT ACCURATELY ACTUAL TIMINGS OBTAINABLE UNDER REAL CONDITIONS. THE MODEL IS STILL IN ITS EXPERIMENTAL STAGES, AND CURRENTLY PROVIDES ONLY A FUNCTIONAL CAPABILITY FOR USERS TO AUGMENT AND CALIBRATE TO THEIR OWN SATISFACTION.

4.1 READ A SEQUENTIAL DATASET OCCUPYING A WHOLE 2314 DISKPACK, AND FORMATTED ONE RECORD PER BLOCK, ONE BLOCK PER TRACK. TIME THE PROCESS.

NOTES:

1. THE HARDWARE CONSISTS OF ONE 2314 DEVICE (I. E. ONE DRIVE, NOT THE WHOLE FACILITY) ON AN IMPLIED CONTROL UNIT CONNECTED TO AN IMPLIED CHANNEL.

2. THERE IS ONE DATASET, "DS", WHICH IS SEQUENTIALLY ORGANIZED, ("S"), AND HAS 4000 RECORDS OF 7294 CHARACTERS EACH. IT IS TO RESIDE ON 2314 DEVICES (IN THIS CASE IT WILL JUST FIT ON ONE 2314 DISKPACK).

THE PROCEDURE SPECIFIES:

3. PRINT THE DATASET PARAMETERS. NOTE THAT A DATASET CONSISTS OF "FILES", WHICH IN TURN CONSIST OF "EXTENTS". IN THIS CASE, "DS" CONSISTS OF ONE FILE, WHICH CONSISTS OF ONE EXTENT, ENCOMPASSING A WHOLE 2314 DISKPACK.

4. READ A RECORD FROM "DS". THE SECOND PARAMETER SPECIFIES A LIST OF RECORDS TO BE READ FROM THE DATASET. AS EACH RECORD IS READ, IT IS REMOVED FROM THE LIST. THE SECOND, OR "LIST" PARAMETER HAS BEEN OMITTED IN THIS CASE, IMPLYING THAT A LIST CONSISTING OF ALL THE RECORDS OF THE FILE IS TO BE ASSUMED BY DEFAULT. THE THIRD PARAMETER "R", SPECIFIES THAT IF THE RECORD IS SUCCESSFULLY READ (I. E. AS LONG AS THE LIST IS NOT EMPTY AND AN END-OF-DATA IS NOT REACHED ON "DS"), THE NEXT STEP OF THE PROCEDURE TO BE EXECUTED IS THE ONE LABELLED "R". WHEN "READS" FAILS, EXECUTION PROCEEDS WITH THE NEXT STEP OF THE PROCEDURE.

   THIS STATEMENT WILL, IN EFFECT, CAUSE THE WHOLE DATASET TO BE READ SEQUENTIALLY.

5. PRINT TIMING STATISTICS. "SIMULATED TIME" IS THE AMOUNT OF TIME THE PROCESS WOULD TAKE AS COMPUTED BY THE SIMULATION. "REAL TIME" IS CPU TIME USED BY THE SIMULATION PROGRAMS. "REDUCTION" IS THE RATIO OF THE FORMER TO THE LATTER.

6. AT PROCEDURE TERMINATION, A PROCEDURE STATEMENT TRACE IS PRINTED OUT. IT IS A LIST OF THE STATEMENT NUMBERS OF THE LAST 24 PROCEDURE STATEMENTS EXECUTED.

THE PHASE II DATA MANAGEMENT SIMULATION SYSTEM

```
 *HARDWARE
      DEVICE    2314
      END


 *DATASET
 DS   DATASET   S,4000,7294,2314
      END


 *PROCEDURE
      PRINT     ,(DS)
 R    READS     DS,,R
      PTIME
      END


*EXECUTE
TABLE INTERPRETATION ERRORS:

PROCEDURE EXECUTION:


                                                           DATA SETS
        DATASET                              FILES
                                                                                                             EXTENTS
NAME TYPE NREC RSIZ DEVT
                              NAME TYPE DEVT  RPB  TPC ALLT  ALL BTYP NBUF WV   CH   EXT   RPC   KL
                               BUF  TNR RSIZ
DS   S    4000 7294 2314                                                                                 DEV  CYL1 NCYL LREC
                                        2314    1   20         0 M        2                  20    0
                                 0 4000 7294
                                                                                                                 1  200 4000
```

```
SIMULATED TIME      104974.63 MS.          REAL TIME      1508 MS.          REDUCTION     69.61 TO 1


ENC OF PROCEDURE,      4004 STATEMENTS EXECUTED

LAST 24 STATEMENTS EXECUTED:
   2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    3    4



*END
END OF RUN
```

4.2 DEFINE A PERSONNEL FILE, STORED IN SEQUENTIAL FORM ON A 2314 DISKPACK. READ THE DATASET.

NOTES:

1. THE PERSONNEL FILE IS ORGANIZED AS FOLLOWS:
   EACH EMPLOYEE HAS A MASTER SEGMENT CONTAINING HIS NAME, AGE, AND EMPLOYEE NO. (NO). THESE FIELDS HAVE 20, 2, AND 10 CHARACTERS, RESP. THERE ARE 1000 OF THESE MASTER SEGMENTS (THAT IS, 1000 EMPLOYEES), AND EACH SEGMENT HAS AN ADDITIONAL 10 BYTES NOT ASSIGNED TO FIELDS. ASSOCIATED WITH EACH MASTER SEGMENT IS A LIST OF JOBS (JOB SEGMENT) AND A LIST OF THE EMPLOYEE'S CHILDREN (CHLD SEGMENT). EACH JOB SEGMENT HAS A JOB TITLE (TITL) FIELD AND A SALARY (SAL) FIELD, AND EACH CHILD SEGMENT HAS A NAME, AGE, AND SEX FIELD. THERE IS AN AVERAGE OF 3 JOB SEGMENTS PER MASTER SEGMENT, AND 2 CHILD SEGMENTS PER MASTER SEGMENT.

2. ALL SEGMENTS HAVE BEEN ASSIGNED TO DATASET "DS", HENCE ALL THE INFORMATION ABOUT AN EMPLOYEE WILL BE STORED IN A SINGLE RECORD OF THAT DATASET, WHICH IS A SEQUENTIAL DATASET RESIDING ON A 2314 DEVICE.

THE PROCEDURE SPECIFIES:

3. PRINT THE SEGMENT AND DATASET TABLES. NOTE THAT EACH SEGMENT IS LISTED TOGETHER WITH ITS FIELDS, AND THAT TOTAL SEGMENT SIZES HAVE BEEN COMPUTED. FURTHERMORE, WHEN THE DATASET PARAMETERS ARE PRINTED OUT, TOTAL (AVERAGE) RECORD SIZE HAS BEEN COMPUTED FROM THE SEGMENT SIZES (42+3*20+2*26=154). NOTE ALSO FROM THE EXTENT TABLE THAT "DS" REQUIRES TWO CYLINDERS OF 2314 STORAGE.

4. READ THE WHOLE DATASET, PRINT TIMING STATISTICS.

THE PHASE II DATA MANAGEMENT SIPULATION SYSTEM

```
*HARCWARE
     DEVICE     2314
     END



*SEGMENTS
MAST SEGMENT    10,,DS,1000
NAME      FIELD 20
AGE       FIELD 2
NU        FIELD 10
JOB   SEGMENT   5,MAST,DS,3
TITL      FIELD 10
SAL       FIELD 5
CHLD  SEGMENT   3,MAST,DS,2
NAME      FIELD 20
SEX       FIELD 1
AGE       FIELD 2
     END



*DATASET
DS   DATASET    S,,,2314
     END



*PROCEDURE
     PRINT      ,(SG,DS)
R    REACS      DS,,R
     PTIME
     END
```

```
*EXECUTE
TABLE INTERPRETATICN ERRORS:

PRCCEDURE EXECUTION:



                                        SEGMENT TABLES

                        SEGMENTS                                                FIELDS


NAME SIZE SUP  DS   KPSS
                                                                  NAME SIZE DIST TYPE SIDS
MAST   42      CS   100C
                                                                  NAME   20
                                                                  AGE     2
                                                                  NO     10
JOB    20 MAST DS      3
                                                                  TITL   10
                                                                  SAL     5

CHLD   26 MAST DS      2                                          NAME   20
                                                                  SEX     1
                                                                  AGE     2



                                                                  DATA SETS

         CATASET                                  FILES                                                              EXTENTS


NAME TYPE NREC RSIZ DEVI
                                   NAME TYPE DEVI  RPB  TPC ALLT  ALL BTYP NBUF NV   CH   EXT   RPC  KL
                                    BUF  TNR RSIZ
                                                                                                           DEV  CYL1 NCYL LREC
DS   S    1000  154 2314
                                             2314   1   20          0 M       2               560    0
                                       C 1000  154
                                                                                                                  1    2 1000


SIMULATED TIME          916.73 MS.          REAL TIME          390 MS.          REDUCTION        2.35 TO 1



END CF PROCEDURE,       1004 STATEMENTS EXECUTED

LAST 24 STATEMENTS EXECUTED:
   2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     3     4

*END
END CF RUN
```

4.3 SPECIFY TWO INDEXED-SEQUENTIAL DATASETS, ONE WITH CYLINDER-EMBEDDED OVERFLOW, THE OTHER WITH OVERFLOW ON A SEPARATE DEVICE. READ 1000 RANDOM RECORDS FROM EACH DATASET AND TIME.

NOTES:

1. THE HARDWARE CONSISTS OF SEVEN 2314 DISKS ON ONE CHANNEL.

2. TWO DATASETS "DS1", AND "DS2" ARE SPECIFIED. THEY EACH HAVE 112200 80-CHARACTER RECORDS, AND ARE TO RESIDE ON 2314 DEVICES.

   THE ACCESS-METHOD-RELATED PARAMETERS SPECIFY:

   (1) THE PRIME DATA AREA IS OCCUPIED TO 1.1 TIMES ITS CAPACITY: THAT IS, THE PRIME AREA IS FULL, AND AN EQUIVALENT OF 10% OF THE PRIME RECORDS IS STORED IN OVERFLOW. THIS MEANS THAT OF THE TOTAL NUMBER OF RECORDS IN THE DATASET (112200), 102000 ARE STORED ON PRIME TRACKS, AND 10200 ARE STORED ON OVERFLOW TRACKS.

   (2) MASTER INDEXES ARE TO BE CREATED WHEN LOWER LEVEL INDEXES OCCUPY MORE THAN TWO TRACKS.

   (3) DISTRIBUTION "CLD" IS SPECIFIED AS THE DISTRIBUTION OF THE OVERFLOW CHAIN LENGTHS. IT SPECIFIES THAT 60% OF THE TRACKS THAT HAVE OVERFLOW RECORDS HAVE ONLY ONE, 30% HAVE TWO, AND 10% HAVE THREE.

3. "DS1" HAS NO FILES EXPLICITLY SPECIFIED, SO ITS PRIME (PR), TRACK INDEX (TI), OVERFLOW (OF), CYLINDER INDEX (CI) AND MASTER INDEXES (MI1, MI2, MI3), IF NEEDED, WILL BE SPECIFIED BY DEFAULT, AND WILL RESIDE ON 2314 DEVICES. "PR", "TI", AND "OF" FILES WILL SHARE CYLINDERS, AND RECORD LENGTHS FOR THE INDEX FILES WILL BE 10 BYTES. KEY LENGTH HAS NOT BEEN SPECIFIED ON THE "PARAM" CARD, SO IT IS TAKEN BY DEFAULT TO BE 10 BYTES.

4. "DS2" DIFFERS FROM "DS1" IN THAT A SPECIFIC ASSIGNMENT OF THE OVERFLOW FILE HAS BEEN MADE TO DEVICE "DEV5", THUS PROVIDING A SEPARATE OVERFLOW AREA.

5. A LIST CALLED "LIST" IS PROVIDED FOR USE BY THE PROCEDURE. IT IS SPECIFIED TO BE A RANDOM LIST OF 1000 INTEGER VALUES, TO BE CHOSEN FROM THE RANGE 1-112200 (THE RANGE OF RECORD NUMBERS ON "DS1" AND "DS2").

THE PROCEDURE SPECIFIES:

6. PRINT THE DATASET PARAMETERS. NOTE THE DEFAULT SPECIFICATIONS FOR PRIME, CYLINDER INDEX, TRACK INDEX AND (FOR "DS1") OVERFLOW FILES THAT HAVE BEEN SUPPLIED BY THE SYSTEM.

7. RESET A TIMER TO ZERO.

8. USING RANDOM ACCESSING, READ FROM "DS1" THE RECORDS SPECIFIED BY LIST "LIST". PRINT TIMING STATISTICS.

9. INITIALIZE "LIST" TO ITS ORIGINAL STATE AS DEFINED IN THE LIST SPECIFICATION.

10. RESTORE THE SYSTEM TO TIME 0.

11. READ FROM "DS2" AND TIME. NOTE THAT THE TIME TO READ FROM "DS2" IS NOT SIGNIFICANTLY DIFFERENT FROM THE TIME TAKEN TO READ FROM "DS1". THE EXTRA TIME NEEDED FOR "DS2" TO PERFORM OVERFLOW SEEKS (ITS OVERFLOW RECORDS ARE SPREAD RANDOMLY OVER TWELVE CYLINDERS) IS OFFSET BY THE FACT THAT THE PRIME RECORDS OF "DS2" ARE SPREAD OVER ONLY 179 CYLINDERS, AS OPPOSED TO 200 FOR "DS1".

```
                    THE PHASE II DATA MANAGEMENT SIMULATION SYSTEM



*HARCWARE
DEV1 DEVICE     2314
DEV2 DEVICE     2314
DEV3 DEVICE     2314
DEV4 DEVICE     2314
DEV5 DEVICE     2314
DEV6 DEVICE     2314
DEV7 DEVICE     2314
     END



*DATASET
DS1  DATASET    IS,112200,80,2314
      PARAM     1.1,2,,CLD
DS2  DATASET    IS,11220C,80,2314
      PARAM     1.1,2,,CLD
       FILE     OF
        EXTENT DEV5
     END



*DISTRIBUTIONS
CLD  DIST       D,I
                         1,.6C
                         2,.30
                         3,.10
     END



*LISTS
LIST LIST       RL,1,1000,1,112200
     END
```

```
*PROCEDURE
      PRINT       ,(DS)
      TIME
X     READR       DS1,LIST,X
      PTIME
      INIT        LIST
      RESTORE
Y     READR       DS2,LIST,Y
      PTIME
      END



*EXECUTE
TABLE INTERPRETATION ERRORS:

PROCEDURE EXECUTION:
```

```
                                                              DATA SETS
        DATASET                           FILES                                                             EXTENTS

NAME TYPE      NREC RSIZ
DEVT
                          NAME TYPE DEVT  RPB  TPC ALLT  ALL BTYP NBUF WV   CH   EXT   RPC   KL
                          BUF        TNR RSIZ                                                       DEV  CYL1 NCYL       LREC

DS1  IS      112200   8C
2314
     PPF   NMI   KL CLD
   1.100    2    10 CLD
                              PR    2314    1   17         0 M      2                510   10
                          0      102000    80                                                       DEV1    1  200     102000
                              OF    2314    1    2         0 M      2                 51   10
                          0       10200    90                                                       DEV1    1  200      10200
                              TI    2314    1    1         0 M      2                 34   10
                          0        6800    10                                                       DEV1    1  200       6800
                              CI    2314    1   20         0 M      2                680   10
                          0         200    10                                                       DEV2    1    1        200
                              MI1   2314    1   20         0 M      2                880   10
                          0           5    10
DS2  IS      1122CC   8C                                                                            DEV3    1    1          5
2314
```

```
         PPF   NMI    KL CLD
        1.100    2    10 CLD
                                      OF    2314     1    20          0 M         2                580   10
                                   0       10200    90
                                                                                                                   DEV5    1   18       10200
                                      PR    2314     1    19          0 M         2                570   10
                                   0      102000    80
                                                                                                                   DEV4    1  179      102030
                                      TI    2314     1     1          0 M         2                 38   10
                                   0        6800    10
                                                                                                                   DEV4    1  179        6800
                                      CI    2314     1    20          0 M         2                880   10
                                   0         179    10
                                                                                                                   DEV6    1    1         179
                                      MI1   2314     1    20          0 M         2                880   10
                                   0           5    10
                                                                                                                   DEV7    1    1           5

   SIMULATED TIME        12677C.56 MS.          REAL TIME        2236 MS.          REDUCTION        56.70 TO 1

   SIMULATED TIME        126640.25 MS.          REAL TIME        2184 MS.          REDUCTION        57.99 TO 1


   END CF PROCEDURE,       2009 STATEMENTS EXECUTED

   LAST 24 STATEMENTS EXECUTEC:
      7    7    7    7    7    7    7    7    7    7    7    7    7    7    7    7    7    7    7    7    7    7    8    9


   *END
   END OF RUN
```

4.4 CONSTRUCT TWO SEQUENTIAL DATASETS THAT SHARE CYLINDERS ON A 2314 DEVICE. CONSTRUCT TWO MORE DATASETS WITH THE SAME CHARACTERISTICS BUT OCCUPYING SEPARATE EXTENTS OF A 2314 DEVICE. MERGE THE FIRST TWO DATASETS, TIME, THEN MERGE THE SECOND TWO DATASETS AND TIME.

NOTES:

1. THREE 2314 DISK DRIVES ON TWO CHANNELS ARE SPECIFIED.

2. "DS1" IS SPECIFIED AS A SEQUENTIAL DATASET WITH FULL-TRACK (7294 BYTES) RECORDS TO BE STORED ON 2314 DEVICES. ITS ASSOCIATED FILE, "FL1" IS ALLOCATED ACROSS 200 CYLINDERS, OCCUPYING 15 TRACKS ON EACH CYLINDER. THE NUMBER OF RECORDS OF "DS1" IS NOT SPECIFIED, BUT IS TO BE INFERRED FROM THE AMOUNT OF SPACE ALLOCATED.

3. "DS2" IS ANOTHER SEQUENTIAL DATASET. ITS ASSOCIATED FILE SHARES EXTENTS WITH "FL1", OCCUPYING FIVE TRACKS PER CYLINDER.

4. "DS3" AND "DS4" ARE DATASETS SIMILAR TO "DS1" AND "DS2", DIFFERING ONLY IN THAT INSTEAD OF SHARING CYLINDERS ACROSS A DEVICE, THEY OCCUPY FULL CYLINDERS IN SEPARATE EXTENTS ON THE SAME DEVICE.

5. "DS5" IS AN OUTPUT DATASET FOR THE MERGE OPERATIONS.

THE PROCEDURE SPECIFIES:

6. PRINT DATASET PARAMETERS. NOTE THAT THE NUMBER OF RECORDS IN EACH DATASET HAS BEEN COMPUTED FROM FILE AND EXTENT PARAMETERS.

7. "SYNCHRONIZE" OPERATIONS AT LABELS "RD1" AND "RD2" IN THE SENSE THAT A TRANSFER TO "RD1" OR "RD2" WILL RESULT IN AN "INDETERMINATE TRANSFER" TO ONE OF THE LABELS, ON THE BASIS THAT THE LISTS SPECIFIED AT THE TWO LABELS SHOULD BE EXHAUSTED AT APPROXIMATELY THE SAME TIME. THIS, IN EFFECT, SIMULATES MERGE-TYPE OPERATIONS BY INTERLEAVING READS ON "DS1" AND "DS2", BUT IN A RANDOM FASHION. THE SPECIFICATION "END1" INDICATES THAT WHEN BOTH LISTS ARE EMPTY, CONTROL TRANSFERS TO LABEL "END1".

8. THE "READS", "READS", "WRITS" SEQUENCE SPECIFIES A SEQUENTIAL READ FROM "DS1" OR "DS2", FOLLOWED BY A SEQUENTIAL WRITE TO "DS5". THIS SEQUENCE IS REPEATED UNTIL "DS1" AND "DS2" HAVE BEEN READ IN THEIR ENTIRETY.

9. PRINT TIMING STATISTICS, RESTORE THE SYSTEM TO TIME 0., AND REPEAT WITH DATASETS "DS3" AND "DS4".

10. NOTE THAT THE SECOND OPERATION TAKES CONSIDERABLY LONGER THAN THE FIRST, AS EXPECTED, SINCE A SIGNIFICANT AMOUNT OF ARM MOVEMENT TAKES PLACE WHEN MOVING BETWEEN "DS3" AND "DS4".

THE PHASE II DATA MANAGEMENT SIMULATION SYSTEM

```
*HARDWARE
DEVA DEVICE     2314
DEVB DEVICE     2314
     CHANNEL
DEVC DEVICE     2314
     END



*DATASETS
DS1  DATASET    S,,7294,2314
FL1   FILE      ,,,15,CYL,200
DS2  DATASET    S,,7294,2314
      FILE      ,,,5,CYL,200,,,,,FL1
DS3  DATASET    S,,7294
      FILE
       EXTENT   DEVB,1,150
DS4  DATASET    S,,7294
      FILE
       EXTENT   DEVB,151,50
DS5  DATASET    S,4000,7294
      FILE
       EXTENT   DEVC
     END



*PROCEDURE
     PRINT      ,(DS)
     SYNC       END1,(RC1,RD2)
     SYNC       END2,(RC3,RD4)
RD1  REACS      DS1,,WR1
RD2  REACS      DS2,,WR1
WR1  WRITS      DS5,,RD1
END1 PTIME
     RESTORE
RD3  REACS      DS3,,WR2
RD4  REACS      DS4,,WR2
WR2  WRITS      DS5,,RD3
END2 PTIME
     END
```

```
*EXECUTE
TABLE INTERPRETATION ERRORS:

PROCEDURE EXECUTION:


                                                   DATA SETS
                                                                                         EXTENTS
        DATASET                      FILES

NAME TYPE NREC RSIZ DEVT
                           NAME TYPE DEVT  RPB  TPC ALLT  ALL BTYP NBUF NV   CH   EXT   RPC   KL
                            BUF  TNR RSIZ                                                          DEV  CYL1 NCYL LREC
DS1  S    3000 7294 2314
                           FL1       2314   1   15 CYL   200 M     2                    15    0
                             0 3000 7294                                                           DEVA    1  200 3000
DS2  S    1000 7294 2314
                                     2314   1    5 CYL   200 M     2           FL1       5    0
                             0 1000 7294                                                           DEVA    1  200 1000
DS3  S    3000 7294
                                            1   20         0 M     2                    20    0
                             0 3000 7294                                                           DEVB    1  150 3000
DS4  S    1000 7294
                                            1   20         0 M     2                    20    0
                             C 1000 7294                                                           DEVB  151   50 1000
DS5  S    4000 7294
                                            1   20         0 M     2                    20    0
                             C 4000 7294                                                           DEVC    1  200 4030

SIMULATED TIME       173524.63 MS.        REAL TIME       3328 MS.        REDUCTION       52.14 TO 1

SIMULATED TIME       247174.63 MS.        REAL TIME       3328 MS.        REDUCTION       74.77 TO 1


END CF PROCEDURE,     16007 STATEMENTS EXECUTEC

LAST 24 STATEMENTS EXECUTED:
 9  11   9  11   9  11   9  11   9  11  10  11   9  11   9  11  10  11  10  11  10  11  12  13

*END
END OF RUN
```

4.5 DEFINE A HIERARCHICALLY STRUCTURED DATABASE; ASSIGN IT TO THREE DATASETS, TWO SEQUENTIAL AND ONE INDEX-SEQUENTIAL (NO OVERFLOW). DEFINE A SET OF QUALIFICATIONS, AND ACCESS THE DATASETS BASED ON THESE QUALIFICATIONS.

NOTES:

1. THE SEGMENT HIERARCHY AND ASSIGNMENT TO DATASETS IS GRAPHICALLY DESCRIBED IN FIGURE 4.5.1

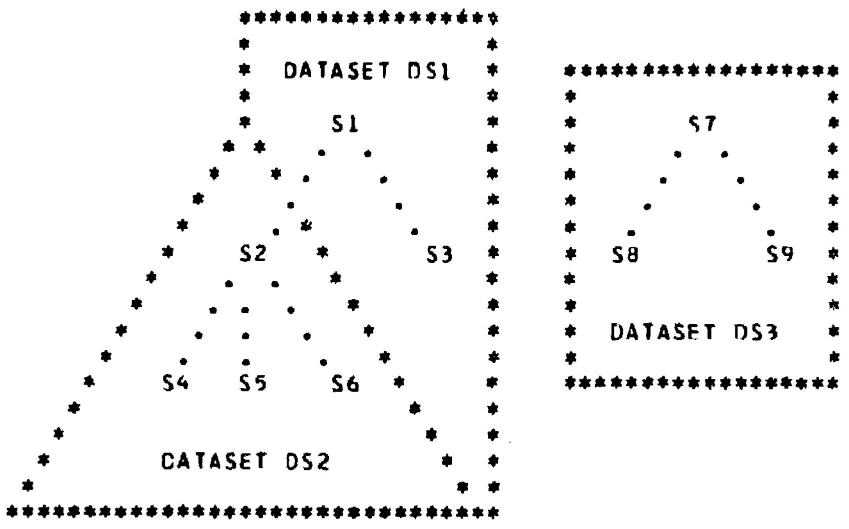


FIGURE 4.5.1

2. SOME OF THE SEGMENTS HAVE FIELDS ASSIGNED TO THEM WHOSE VALUES, IN TURN, ARE CHARACTERIZED BY DISTRIBUTIONS; FOR EXAMPLE, FIELD "4.1" IS CHARACTERIZED BY DISTRIBUTION "A2", WHICH SPECIFIES THAT THE FIELD CONTAINS ONE OF TWO VALUES: "B1", WHICH OCCURS 90% OF THE TIME, AND "B2" WHICH OCCURS 10% OF THE TIME. NOTE THAT UNLIKE "A2", THE OTHER DISTRIBUTIONS HAVE BEEN SPECIFIED IN CUMULATIVE FORM, A USER OPTION.

   FIELDS "2.1" AND "7.1" ARE "SORT" FIELDS. NOTE THAT EACH OF THESE FIELDS IS IN THE HIGHEST LEVEL SEGMENT OF ITS DATASET. THE RECORDS OF DATASETS "DS2" AND "DS3" WILL BE ORDERED ON THESE FIELDS.

3. THE QUALIFICATION SPECIFICATION ILLUSTRATES THE THREE TYPES OF QUALIFICATION STATEMENT:

   "Q1" (A FIELD QUALIFICATION) STATES THAT AN "S2" SEGMENT QUALIFIES BY "Q1" IF FIELD "4.1" HAS VALUE "B2".

"Q3" (A BOOLEAN QUALIFICATION) STATES THAT AN "S2" SEGMENT QUALIFIES BY "Q3" IF IT QUALIFIES BOTH BY "Q1" AND "Q2".

"Q5" (A SEGMENT QUALIFICATION) STATES THAT AN "S1" SEGMENT QUALIFIES IF IT HAS EXACTLY ONE ("EQ,1") "S2" SEGMENT SUBORDINATE TO IT THAT QUALIFIES BY "Q1".

4. FOR ILLUSTRATIVE PURPOSES (THEY ARE NOT USED), TWO LISTS BASED ON QUALIFICATION ARE SPECIFIED. "L1S" IS TO BE A SEQUENTIAL LIST BASED ON QUALIFICATION "Q1"; THAT IS, THE LIST ELEMENTS ARE RECORD NUMBERS OF RECORDS QUALIFIED BY "Q1", AND ARE ARRANGED IN SORT ORDER. "L1R" DIFFERS FROM "L1S" IN THAT THE RECORD NUMBERS ARE ARRANGED IN RANDOM ORDER.

THE PROCEDURE SPECIFIES:

5. PRINT THE SEGMENT TABLES

6. PRINT THE DATASET TABLES. NOTE THAT "DS1", "DS2", AND "DS3" HAVE 100000, 300000, AND 10000 RECORDS, RESPECTIVELY, WHICH ARE THE NUMBER OF SEGMENTS "S1", "S2", AND "S7", RESPECTIVELY. (EACH OF THESE SEGMENTS IS THE HIGHEST LEVEL SEGMENT ON ITS DATA SET). NOTE ALSO THAT DEFAULT "PARAM" PARAMETERS ("PPF", "NMI","KL", AND "CLD") HAVE BEEN SUPPLIED FOR INDEX-SEQUENTIAL DATASET "DS2". AS A RESULT, "DS2" CONSISTS ONLY OF PRIME, TRACK INDEX, AND CYLINDER INDEX FILES.

7. PRINT QUALIFICATION TABLES. IN ADDITION TO THE INPUT INFORMATION, THESE TABLES ALSO SUPPLY SOME RESULTING QUALIFICATION INFORMATION: "LRQ" WHICH IS THE "LOW RECORD QUALIFYING", "HRQ" WHICH IS THE "HIGH RECORD QUALIFYING", AND "NRQ" WHICH IS THE NUMBER OF RECORDS QUALIFYING ON THE INTERVAL ("LRQ","HRQ").

"Q1" IS A FIELD QUALIFICATION ON FIELD "4.1", WHICH IS IN SEGMENT "S4" WHICH RESIDES ON DATASET "DS2" - HENCE "Q1" QUALIFIES 57000 RECORDS ON "DS2" RANDOMLY OCCURRING OVER THE WHOLE DATASET. THIS NUMBER IS ARRIVED AT AS FOLLOWS:

FIELD "4.1" EQUALS "82" 10% OF THE TIME, HENCE 10% OF THE "S4" SEGMENTS QUALIFY. A RECORD OF "DS2" IS PRESUMED TO QUALIFY BY "Q1" IF IT HAS AT LEAST ONE "S4" SEGMENT THAT QUALIFIES. SINCE THERE ARE TWO "S4" SEGMENTS PER RECORD OF "DS2", THE PROBABILITY THAT A RECORD OF "DS2" DOES NOT QUALIFY IS:

(1 - .10)**2

AND THE PROBABILITY THAT IT DOES IS:

1 - (1 - .10)**2 =.19

HENCE THERE ARE (.19)*(300000) = 57000 RECORDS OF "DS2" THAT QUALIFY BY "Q1".

"Q2", "Q3', AND "Q4" ARE ALSO QUALIFICATIONS ON "DS2". "Q5" IS A QUALIFICATION ON "DS1", AND "Q6", "Q7", AND "Q8" ARE QUALIFICATIONS ON "DS3". THE LATTER THREE INVOLVE A SORT FIELD, HENCE QUALIFY ONLY OVER A SUBSET OF THE WHOLE DATASET RECORD RANGE.

"Q4" AND "Q5" ARE NOT USED BY THE PROCEDURE, BUT ARE INCLUDED TO ILLUSTRATE THE "OR" BOOLEAN AND SEGMENT QUALIFICATIONS, RESPECTIVELY.

8. PRINT LIST TABLES. NOTE THE ENTRIES FOR "LIS" AND "LIR". EACH CONSISTS OF 57000 INTEGERS BETWEEN 1 AND 300000, WHICH IS THE SET OF RECORD NUMBERS QUALIFIED BY "Q1". "LIS" IS A SEQUENTIAL LIST ("SQ") AND "LIR" IS A RANDOM LIST ("RQ")

   LIST "//1" IS AN "EMBEDDED" LIST; THAT IS, IT WAS EMBEDDED IN PROCEDURE STATEMENT 1.

   LISTS "//2" AND "//3" REPRESENT LISTS IMPLIED BY PROCEDURE STATEMENTS LABELLED "X" AND "Y", RESPECTIVELY. LIST "//2" IS A SEQUENTIAL LIST, SINCE THE OPERATION AT "X" IS A "READS" (READ SEQUENTIAL), AND IS BASED ON QUALIFICATION "Q8", AS SPECIFIED BY THE MODIFIER AND OBJECT AT "X". SIMILARLY, "//3" IS A RANDOM LIST BASED ON QUALIFICATION "Q3".

9. SET A TIMER TO ZERO, READ (USING SEQUENTIAL ACCESS) THE RECORDS QUALIFIED BY "Q8", AND PRINT TIMING STATISTICS. REPEAT FOR QUALIFICATION "Q3", USING RANDOM ACCESS (ISAM).

10. NOTE THAT 5500 PROCEDURE STATEMENTS HAVE BEEN EXECUTED. THIS NUMBER CAN BE BROKEN DOWN AS FOLLOWS:

2992+2500	TO READ QUALIFIED RECORDS
2	"READS" AND "READR" WHICH FAILED (DUE TO END-OF-LIST)
6	OTHER PROCEDURE STATEMENTS (INCLUDING "END")

11. NOTE THAT THE "READS" TOOK AN AMOUNT OF TIME NEEDED TO READ 114 TRACKS OF 2314 STORAGE (2965 = 25*114 APPROX) WHICH ACCOUNTS FOR READING FROM THE BEGINNING OF DATASET "DS3" DOWN TO RECORD 5000.

```
*HARCWARE
DEV1 DEVICE     2314
DEV2 DEVICE     2314
DEV3 DEVICE     2314
DEV4 DEVICE     2314
UEV5 DEVICE     2314
DEV6 DEVICE     2314
     END


*DATASETS
DS1  DATASET    ,,,2314
DS2  DATAS'I    15,,,2314
DS3  DATASET    ,,,2314
     END


*SEGMENTS
S1    SEGMENT   5,,DS1,100000
S3     SEGMENT  7,S1,DS1,2
S2     SEGMENT  5,S1,DS2,3
2.1        FIELD 4,12,S
S4      SEGMENT 2,S2,DS2,2
4.1        FIELD 5,A2
4.2        FIELD 3,R1
S5      SEGMENT 2,S2,DS2,2
S6      SEGMENT 3,S2,DS2,2
S7    SEGMENT   10,,DS3,1000C
7.1        FIELD 4,R3,S
S8     SEGMENT  5,S7,DS3,5
S9     SEGMENT  5,S7,DS3,5
     END


*DISTRIBUTIONS
A2   DIST      D,A
```

```
                       B1,.90
                       B2,.10
I2    DIST       C,I
                       0,0.
                       100,1.
R3    DIST       C,R
                       0,0.
                       100,1.
R1    DIST       C,R
                       5.,0.
                      10.,.50
                      115.,.75
                      120.,1.
      END



*QUALIFICATIONS
Q1     CUAL      4.1,EQ,B2
Q2     QUAL      4.2,GT,119.
Q3      QUAL     Q1,AND,Q2
Q4      CUAL     C1,OR,Q2
Q5      CUAL     S1,HAS,C1,EQ,1
Q6     QUAL      7.1,LE,50
Q7     CUAL      7.1,GT,25
Q8      QUAL     C6,AND,Q7
      END



*LIST
LIS    LIST      SQ,Q1
LIR    LIST      RQ,Q1
      END



*PROCEDURE
      PRINT      ,(SG,DS,CU,LI)
      TIME
X     READS     C CR,,X
      PTIME
      TIME
Y     READR     C Q3,,Y
      PTIME
      END
```

```
*EXECUTE
TABLE INTERPRETATION ERRORS:

PROCEDURE EXECUTION:


                                        SEGMENT TABLES

                       SEGMENTS                                                          FIELDS


NAME SIZE SUP  DS          NPSS
                                                                  NAME SIZE DIST TYPE STDS
S1      5      DS1       100000
S3      7 S1   DS1            2
S2      9 S1   DS2            3
                                                                  2.1       4 12    S
S4     10 S2   DS2            2
                                                                  4.1       5 A2
                                                                  4.2       3 R1
S5      2 S2   DS2            2
S6      3 S2   DS2            2
S7     14      DS3        10000
                                                                  7.1       4 R3    S
S8      5 S7   DS3            5
S9      5 S7   DS3            5


                                                                       DATA SETS

         DATASET                                     FILES                                                                          EXTENTS


NAME TYPE      NREC RSIZ
DEVT
                                    NAME TYPE DEVT  RPB  TPC ALLT  ALL BTYP NBUF WV   CH   EXT   RPC   KL
                                     BUF        TNR RSIZ
                                                                                                                     DEV  CYL1 NCYL      LREC
DS1  S       100000   19
2314
                                              2314    1   20         0 M      2               1220    0
                                     0     100000   19
                                                                                                                     DEV1    1   82    100000
DS2  IS      300000   39
2314
    PPF        NMI    KL CLD
   1.000  10000000    10
                                       PR     2314    1   19         0 M      2                703   10
                                     0     300000   39
```

```
                                                                                                              DEV2    1   200     140600
                                                                                                              DFV3    1   200     281200
                                 TI    2314    1     1          0 M      2              38    10              DFV4    1    27     300000
                              0       16218   10

                                                                                                              DEV2    1   200       7600
                                                                                                              DFV3    1   200      15200
                                 CI    2314    1    20          0 M      2             880    10              DFV4    1    27      16218
                              0         427   10

DS3  S         10000    64                                                                                    DEV5    1     1        427
2314

                                       2314    1    20          0 M      2             880     0
                              0       10000   64

                                                                                                              DEV6    1    12      10000

                                QUALIFICATION TABLES

NAME  ARG1  REL   ARG2          QREL      NO  LRQ        HRQ        NRQ
Q1    4.1   EQ    B2                       0    1     300000      57000
Q2    4.2   GT    119.                     0    1     300000      29250
Q3    Q1    AND   C2                       0    1     300000       2992
Q4    Q1    OR    C2                       0    1     300000      80692
Q5    S1    HAS   Q1             EQ        1    1     100000      35429
Q6    7.1   LE    50                       0    1       5000       5000
Q7    7.1   GT    25                       0 2501      10000       7500
Q8    Q6    AND   Q7                       0 2501       5000       2500


                                          LISTS

                 PARAMETERS                                                                            CONTENTS

NAME TYPE MO          SIZE        LO        HS DIST      SIZO  ILO       IHS       RLO
       RHS
L1S  SQ   Q1         5700C        1.    300000.         57000    1    300000      0.0
   C.0
L1R  RC   Q1         5700C        1.    300000.         5700C    1    300000      0.0
   0.0
//1  LL   A              4        0.         0.             0    0         0      0.0
   0.0

//2  SQ   Q8          250C     2501.      5000.          2500 2501       5000      0.0       SG    DS    QU    LI
   0.0
//3  RC   Q3          2992        1.    3C000C.          2992    1    300000      0.0
   C.0

SIMULATED TIME        2964.99 MS.        REAL TIME       1950 MS.          REDUCTION      1.52 TO 1

SIMULATED TIME      6C6808.56 MS.        REAL TIME      87412 MS.          REDUCTION      6.94 TO 1
```

```
END CF PROCEDURE,      5500 STATEMENTS EXECUTED

LAST 24 STATEMENTS EXECUTED:
    6    6    6    6    6    6    6    6    6    6    6    6    6    6    6    6    6    6    6    6    6    6    7    8



*END
END OF RUN
```

5.0 ERROR CONDITIONS

EACH ERROR THAT CAN BY DETECTED BY PHASE II HAS A UNIQUE ERROR CODE ASSIGNED TO IT, AND FALLS INTO ONE OF SEVEN ERROR CLASSES. EACH ERROR CLASS IS ASSOCIATED WITH A TYPE OF ERROR, AND DETERMINES WHAT WILL BE THE DISPOSITION OF THE ERROR CONDITION. THE CLASSES ARE:

CLASS 0 -	WARNING - CONTINUE PROCESSING
CLASS 1 -	ERROR DURING PROCEDURE EXECUTION - GO TO ERROR LABEL IN PROCEDURE, IF POSSIBLE. OTHERWISE, DUMP ALL TABLES AND FLUSH DOWN TO NEXT CONTROL CARD.
CLASS 2 -	ERROR DETECTED IN INPUT - FLUSH DOWN TO NEXT CONTROL CARD, DELETE EXECUTION OF THE PROCEDURE
CLASS 3 -	ERROR DETECTED DURING INTERPRETATION OF TABLES - CONTINUE, BUT DELETE EXECUTION OF THE PROCEDURE
CLASS 4 -	SYSTEM ERROR - DUMP ALL TABLES, FLUSH DOWN TO NEXT CONTROL CARD - DELETE EXECUTION OF THE PROCEDURE
CLASS 5 -	CATASTROPHIC ERROR - ABEND
CLASS 6 -	MILD ERROR IN INPUT - CONTINUE, DELETE EXECUTION OF THE PROCEDURE

THE FOLLOWING IS A LIST AND DESCRIPTION OF ALL ERRORS DETECTABLE BY THE SYSTEM:

CODE	DESCRIPTION	CLASS	ROUTINE
1	ATTEMPT TO REOPEN AN ALREADY OPEN FILE	1	OPEN
2	ATTEMPT TO OPEN TOO MANY FILES AT ONCE	1	OPEN
3	ATTEMPT TO CLOSE A FILE THAT IS NOT OPEN	1	CLOSE
4	ATTEMPT TO CLOSE A FILE WHICH HAS I/O REQUESTS PENDING THAT ARE NOT KNOWN TO THE SYSTEM	1	CLOSE
5	ATTEMPT TO OPEN SEQUENTIAL FILE WITH BUFFER TYPE OTHER THAN 'M', 'L' OR BLANK.	1	OPEN
6	NOT ENOUGH DEVICES OF A REQUIRED TYPE TO SATISFY ALL SPACE REQUESTS	3	ALLOC
7	DEVICE REFERRED TO BY FILE DOES NOT EXIST	3	ALLOC
8	FILES SHARE CYLINDERS, BUT THEIR REQUIREMENTS DO NOT		

	MATCH	3	ALLOC
9	REQUESTED RECORD NOT IN FILE	1	LOCATE
10	NO DEVICE CLASS ASSIGNED TO FILE	3	IFL
11	FILE REFERRED TO BY FILE WITH WHICH IT SHARES EXTENTS DOES NOT EXIST	3	IFL
12	DEVICE ASSIGNED TO FILE DOES NOT EXIST	3	IFL
13	DEVICE CLASS ASSIGNED TO FILE DOES NOT EXIST	3	IFL
14	UNACCEPTABLE LIST INPUT: (1) LIST TABLE FULL (2) LITERAL LIST CONTENTS TABLE FULL (3) LITERAL LIST CONTENTS KEYWORD NOT BLANK (4) INCORRECT KEYWORD	2	RLI
15	ATTEMPT TO USE AN ILLEGALLY SPECIFIED LIST	1	GET
16	SAME	1	IGET
17	A REAL OR ALPHAMERIC LIST HAS BEEN SPECIFIED - AN INTEGER LIST IS REQUIRED, OR VICE-VERSA	1	IGET GET
18	LIST TYPE INCORRECTLY SPECIFIED	0	ILI
19	LIST INCORRECTLY SPECIFIED: (1) ALPHAMERIC LIST HAS BEEN SPECIFIED AS SEQUENTIAL (2) RANDOM/SEQUENTIAL LIST HAS BEEN SPECIFIED AS REAL OR ALPHAMERIC (3) RANDOM/SEQUENTIAL LIST HAS BEEN SPECIFIED WITH MORE ENTRIES THAN ITS INTERVAL PROVIDES	0	ILI
20	LIST TABLE FULL	6	CREATL
21	ATTEMPT TO DEFINE TOO MANY CREATED LISTS	6	CREATL
22	ATTEMPT TO "PUT" INTO LIST WITH ILLEGAL MODE	4	PUT
23	OUT OF LITERAL LIST ENTRY SPACE	2	PUT
24	DISTRIBUTION SPECIFIED FOR RANDOM LIST DOES NOT EXIST	0	ILI
25	NON-UNIFORM DISTRIBUTION SPECIFIED FOR RANDOM/SEQUENTIAL LIST	0	ILI
26	ILLEGAL "PLT" - LIST ALREADY CREATED	4	PUT
27	ATTEMPT TO READ BEYOND END-OF-DATA	1	READS

28	ATTEMPT TO WRITE TOO MANY RECORDS TO FILE	1	WRITS
29	ILLEGAL I/O REQUEST:	1	
	(1) READ ON FILE OPENED FOR WRITE		READS
	(2) WRITE ON FILE OPENED FOR READ		WRITS
	(3) UPDATE ON CLOSED FILE, OR ONE NOT OPEN FOR READ IN LOCATE MODE	1	UPDATS
30	INPUT LINE EXCEEDS 130 CHARACTERS	2	INTERP
31	MASTER HAS ILLEGAL TYPE FIELD	4	INTERP
32	ILLEGAL CHARACTER IN INPUT FIELD, OR ALPHAMERIC FIELD EXCEEDS FOUR CHARACTERS	6	INTERP
33	PROCEDURE HAS TOO MANY STEPS	2	RPR
34	UNRECOGNIZED OPERATION IN PROCEDURE	0	RPR
35	TOO MANY UNRECOGNIZED OPERATIONS IN PROCEDURE - APPARENTLY WHAT IS BEING READ IS NOT A PROCEDURE	2	RPR
36	INPUT/OUTPUT QUEUE TABLE OVERFLOW - TOO MANY PENDING I/O REQUESTS WITHOUT INTERVENING WAITS.	1	AC
37	PENDING I/O REQUEST IN BUFFER NOT RECOGNIZED BY SYSTEM	4	RESET
38	ERROR IN DISTRIBUTION INPUT: (1) TOO MANY DISTRIBUTIONS SPECIFIED (2) TOO MANY ENTRIES IN DISTRIBUTION CONTENTS TABLE (3) INCORRECT KEYWORD	2	RDI
39	DISTRIBUTION INCORRECTLY SPECIFIED: (1) MODE NOT A, R, OR I (2) TYPE NOT D OR C (3) FIRST VALUE OF CONTINUOUS DISTRIBUTION NOT 0. (4) DISTRIBUTION DOES NOT ACCUMULATE TO 1.	0	RDI
40	ALPHAMERIC DISTRIBUTION SPECIFIED AS "C"; "D" SUBSTITUTED	0	RDI
41	ATTEMPT TO USE AN INCORRECTLY SPECIFIED DISTRIBUTION. ROUTINES: DISTV,DISTC,DISTA,IDISTA,DIST,IDIST	1	
42	ILLEGAL USE OF IDIST FUNCTION: (1) REAL OR ALPHAMERIC DISTRIBUTION - INTEGER DISTRIBUTION REQUIRED (2) LOW OR HIGH VALUES DO NOT MATCH DEFINED VALUES OF DISCRETE DISTRIBUTION (3) LOW OR HIGH VALUES NOT IN RANGE OF INTERPOLATE DISTRIBUTION	1	IDIST
43	SYSTEM ERROR - DISCRETE DESTRIBUTION	4	IDIST

44	SYSTEM ERROR - CONTINUOUS DISTRIBUTION	4 IDIST
45	SYSTEM ERROR - DISCRETE DISTRIBUTION	4 DIST
46	SYSTEM ERROR - CONTINUOUS DISTRIBUTION	4 DIST
47	ILLEGAL USE OF DIST FUNCTION: (1) INTEGER DISTRIBUTION - REAL OR ALPHAMERIC REQUIRED (2)-(3) AS IN 42	1 DIST
48	NOT USED	
49	GIVEN ARGUMENT DOES NOT MATCH A DEFINED ARGUMENT FOR AN ALPHAMERIC DISTRIBUTION	1 DISTV
50	GIVEN ARGUMENT DOES NOT MATCH A DEFINED ARGUMENT FOR AN INTEGER DISTRIBUTION	1 DISTV
51	ATTEMPT TO USE DISTV (FOR DISCRETE DISTRIBUTIONS) ON A CONTINUOUS DISTRIBUTION	1 DISTV
52	GIVEN ARGUMENT DOES NOT MATCH A DEFINED ARGUMENT FOR A REAL/DISCRETE DISTRIBUTION	1 DISTV
53	SAME AS 49	1 DISTC
54	SAME AS 50	1 DISTC
55	GIVEN ARGUMENT DOES NOT MATCH A TABULATED ARGUMENT FOR A REAL/DISCRETE DISTRIBUTION, OR DOES NOT FALL WITHIN THE RANGE OF A CONTINUOUS DISTRIBUTION	1 DISTC
56	DISTRIBUTION REFERENCED DOES NOT HAVE INTEGER VALUES OR GIVEN VALUE OUT-OF-RANGE	1 IDISTA
57	DISTRIBUTION REFERENCED DOES NOT HAVE REAL OR ALPHA-MERIC VALUES, OR GIVEN VALUE IS OUT-OF-RANGE	1 DISTA
58	ERROR IN HARDWARE INPUT: (1) TOO MANY CHANNELS, CONTROL UNITS, OR DEVICES SPECIFIED (2) TOO MANY CONTROL UNITS ASSIGNED TO ONE CHANNEL (3) TOO MANY CHANNELS ASSIGNED TO ONE CONTROL UNIT (4) INCORRECT KEYWORD (5) TOO MANY DEVICES ASSIGNED TO ONE CONTROL UNIT	2 RHD
59	ERROR IN DEVICE CLASS INPUT: (1) TOO MANY DEVICE CLASSES (2) INCORRECT KEYWORD	2 RDP
60	TABLE REFERRED TO BY DEVICE CLASS ENTRY DOES NOT EXIST	3 IDP
61	DEVICE CLASS REFERRED TO BY DEVICE ENTRY DOES NOT	

	EXIST	3 IDV
62	ERROR IN TABLE INPUT: (1) TOO MANY TABLE ENTRIES (2) TOO MANY TABLE CONTENTS ENTRIES (3) INCORRECT KEYWORD	2 RTB
63	ERROR ON CALL TO TABLE-LOOK-UP: (1) TABLE ARGUMENT TYPE OR VALUE TYPE NOT I, R, OR A. (2) FUNCTION TYPE NOT I OR S	1 TABLE ITABLE
64	FOR A FUNCTION WHOSE VALUES OR ARGUMENTS ARE ALPHAMERIC, THE GIVEN ARGUMENT DOES NOT MATCH A TABULATED ARGUMENT	1 TABLE ITABLE
65	ERROR IN DATASET INPUT: (1) TOO MANY DATASETS, FILES, OR EXTENTS (2) TOO MANY FILES ASSOCIATED WITH ONE DATASET (3) UNDEFINED DATASET TYPE (4) PARAM CARD SUPPLIED WITH SEQUENTIAL DATASET (5) INCORRECT KEYWORD	2 RDS
66	INVALID DATASET TYPE - DATASET PRINT OR DUMP	0 PDS DDS
67	INVALID DATASET TYPE	3 IDS
68	ATTEMPT TO EXECUTE PROCEDURE STATEMENT HAVING A PREVIOUSLY DETECTED ERROR	1 EXPR
69	INVALID ERROR CODE	5 MAIN
70	CONTROL CARD DOES NOT HAVE *, OR HAS ILLEGAL KEYWORD	2 MAIN
71	SYSTEM CANNOT FIND "SYNC" OP ASSOCIATED WITH A SET OF SYNCHRONIZED OPS	4 EXPR
72	ILLEGAL OP NUMBER IN PROCEDURE	4 EXPR IPR
73	SAME	4 AUXPRI AUXPRF
74	NOT USED	
75	LIST REFERRED TO IN PROCEDURE NOT FOUND	0 IPR
76	LABEL REFERRED TO BY PROCEDURE NOT FOUND	0 IPR
77	LABEL REFERRED TO BY "SYNC" NOT FOUND	1 EXPR
78	ERROR IN SEGMENT INPUT: (1) TOO MANY SEGMENTS OR FIELDS (2) INCORRECT KEYWORD	2 RSG

79	REFERENCED SUPERIOR SEGMENT DOES NOT EXIST	3 ISG
80	DATASET REFERRED TO BY SEGMENT DOES NOT EXIST	3 ISG
81	NUMBER-PER-SUPERIOR-SEGMENT FIELD IN SEGMENT INPUT IS ZERO	3 ISG
82	A SEGMENT IS SUPERIOR TO ITSELF	3 ISG
83	DISTRIBUTION REFERRED TO BY FIELD CANNOT BE FOUND	3 IFD
84	DATASET ASSOCIATED WITH SECONDARY INDEX CANNOT BE FOUND	3 IFD
85	SEGMENTS ON SAME DATASET DO NOT HAVE THE SAME DATASET MASTER SEGMENT	3 ISG
86	A KEY FIELD IS IN A SEGMENT WHICH IS NOT A DATASET MASTER	3 ISG
87	THERE ARE MORE THAN ONE SEQUENTIAL KEY FIELDS ON THE SAME DATASET	3 ISG
88	ILLEGAL TYPE IN MASTER - SYSTEM ERROR WHICH RESULTS ONLY IN GARBLED OUTPUT	0 HEAD
89	ERROR IN QUALIFICATION INPUT: (1) TOO MANY QUERIES (2) INCORRECT KEYWORD	2 RQU
90	ILLEGAL TYPE - CONVERT ARGUMENT	4 CONVER
91	ILLEGAL CHARACTER IN NUMERIC VALUE	3 CONVER
92	ILLEGAL RELATIONAL OPERATOR IN QUALIFICATION	3 IQU
93	FIELD REFERRED TO BY QUALIFICATION DOES NOT EXIST	3 IQU
94	SEGMENT REFERRED TO BY QUALIFICATION DOES NOT EXIST	3 IQU
95	QUALIFICATION REFERRED TO BY QUALIFICATION DOES NOT EXIST	3 IQU
96	QUALIFICATIONS FORM A CIRCULAR DEFINITION	3 IQU
97	QUALIFICATIONS REFERRED TO BY A BOOLEAN QUALIFICATION DO NOT QUALIFY THE SAME SEGMENT	3 IQU
98	"SEG" AND THE SEGMENT QUALIFIED BY "Q3" ARE NOT LINEALLY RELATED. SEE DESCRIPTION OF SEGMENT QUALIFICATION.	3 IQU
99	THE "NO" PARAMETER IS IMPROPERLY SPECIFIED IN A SEGMENT QUALIFICATION. IT MUST BE A NON-NEGATIVE INTEGER NOT GREATER THAN THE NUMBER OF SEGMENTS	

	"SEG" PER SEGMENT QUALIFIED BY "Q3". SEE DESCRIPTION OF SEGMENT QUALIFICATION, SECTION 3.3.		
100	QUALIFICATION REFERRED TO BY LIST (BASED ON QUALIFICATION) DOES NOT EXIST.	3	ILI
101	TOO MANY ISAM DATASETS	6	RXIS
102	SAME	3	IXIS
103	NOT USED		
104	CONTRADICTORY PRIME, OVERFLOW, AND TRACK INDEX PARAMETERS SPECIFIED	3	IXIS
105	RANDOM ACCESS TO NON-RANDOM DATASET	1	READR
106	SAME	1	WRITR
107	NO SPACE LEFT IN FILE TABLE FOR FILE CREATED BY DEFAULT	3	CREATE
108	A FILE REQUIRES TOO MANY EXTENTS	3	ALLOC
109	OVERFLOW-CHAIN-LENGTH-DISTRIBUTION REFERRED TO BY ISAM PARAMETER LIST DOES NOT EXIST	3	IXIS
110	DATASET REFERRED TO BY ACCESS OP IN PROCEDURE DOES NOT EXIST	0	IPR
111	SAME FOR FIELD	0	IPR
112	SAME FOR QUALIFICATION	0	IPR
113	ACCESS OP IN PROCEDURE HAS ILLEGAL MODIFIER FIELD	0	IPR
114	NUMBER OF BUFFERS IN AN "OPEN" ACCESS OP IS ILLEGAL	1	EXPR
115	INCORRECT LOGICAL FILE FOR OUTPUT IN PRINT, DUMP, OR TRACE PROCEDURE STATEMENT	0	IPR
116	NO LIST SPECIFIED IN PRINT, DUMP, OR TRACE PROCEDURE STATEMENT	0	IPR
117	TOO MANY TIMERS REQUIRED BY PROCEDURE	0	IPR
118	ERROR LABEL SPECIFIED BY "ERROR" PROCEDURE STATEMENT DOES NOT EXIST	0	IPR
119	ATTEMPT TO UPDATE A DATASET OF A TYPE NOT UPDATABLE	1	EXPR
120	ATTEMPT TO CREATE TOO MANY FILES IN ONE DATASET	3	IXIS
121	EXTENT BOUNDS INCOMPATIBLE WITH NO. OF CYLINDERS ON THE DEVICE	3	ALLOC

122	DISTRIBUTION TABLE FULL	3 CREATD
123	ATTEMPT TO CREATE TOO MANY DEFAULT DISTRIBUTIONS	3 CREATD
124	ATTEMPT TO "PUTD" INTO DISTRIBUTION WILL INVALID MODE SPECIFICATION	4 PUTD
125	OUT OF DISTRIBUTION CONTENTS SPACE	3 PUTD
126	ILLEGAL "PUTD" - DISTRIBUTION ALREADY CREATED	4 PUTD
127	RESERVED	
128	RESERVED	
129	RESERVED	
130	RESERVED	
131	RESERVED	
132	RESERVED	
133	RESERVED	
134	NUMBER OF OUTSTANDING REQUESTS ON A DIRECT ACCESS DATASET EXCEEDS THE NUMBER OF BUFFERS AVAILABLE	1 READD
135	SAME	1 WRITD
136	A WAIT HAS BEEN ISSUED ON A FILE WHICH IS NOT OPEN	1 WAITD
137	RESERVED	
138	RESERVED	

6.C MODEL SIZE LIMITATIONS

THE FOLLOWING ARE THE BUILT-IN LIMITATIONS TO THE SYSTEM DUE TO TABLE SIZES:

ELEMENT	MAXIMUM NUMBER
OPEN FILES	20
LISTS	30
DEFAULT CREATED LISTS	20
LITERAL LIST CONTENTS VALUES	400
STEPS IN PROCEDURE	50
PENDING I/O REQUESTS	20
DISTRIBUTIONS	30
DISTRIBUTION (ARG,VAL) PAIRS	400
CHANNELS	8
CONTROL UNITS	10
DEVICES	30
CONTROL UNITS ON ONE CHANNEL	10
CHANNELS ATTACHED TO ONE CONTROL UNIT	4
TABLES	20
TABLE (ARG,VAL) PAIRS	300
DATASETS	20
FILES	30
EXTENTS	200
FILES PER DATASET	10
SEGMENTS	20
FIELDS	80
QUALIFICATION SPECIFICATIONS	30
TIMERS	10
DEVICES PER CONTROL UNIT	10
DEFAULT CREATED DISTRIBUTIONS	20

7.0 ACCESS METHODS

EACH DATASET TYPE (E. G., "S", "D", "IS") HAS A CORRESPONDING "ACCESS METHOD" WHICH IS EMBODIED AS A PROGRAM, AND DESCRIBES ACCESSING OPERATIONS ON THE DATASET. FOR EXAMPLE, A "SEQUENTIAL" OR "S" TYPE DATASET IS ACCESSED USING THE "SEQUENTIAL ACCESS METHOD".

EACH DATASET CONSISTS OF ONE OR MORE "ELEMENTARY FILES" (HEREINAFTER CALLED "FILES"), AND IT IS THE RESPONSIBILITY OF THE ACCESS METHOD TO RELATE THESE FILES, AND DESCRIBE THE OPERATIONS WHICH MUST BE PERFORMED ON THEM IN ORDER TO RETRIEVE A GIVEN "LOGICAL RECORD" OF THE DATASET. FOR EXAMPLE, A REQUEST FOR RECORD 123 OF AN INDEXED DATASET MIGHT REQUIRE ACCESSES TO A CYLINDER INDEX FILE, A TRACK INDEX FILE, AND THE PRIME DATA FILE IN ORDER TO SATISFY THE REQUEST.

THE FILES OF A DATASET CAN BE SPECIFIED BY THE MODELER AS PART OF HIS MODEL INPUT, OR CAN BE SUPPLIED BY THE ACCESS METHOD TO THE EXTENT THAT IT IS PROGRAMMED TO PROVIDE DEFAULT FILES.

THE THREE ACCESS METHODS DESCRIBED HEREIN HAVE BEEN INCLUDED TO ILLUSTRATE A RANGE OF "SOPHISTICATION" OF ACCESS METHODS. THEY ARE INTENTIONALLY SIMILAR TO THREE IBM OS/360 ACCESS METHODS: (1) BASIC DIRECT ACCESS METHOD (BDAM),(2) QUEUED SEQUENTIAL ACCESS METHOD (QSAM), AND (3) BASIC INDEX SEQUENTIAL ACCESS METHOD (BISAM). THEY REQUIRE 165, 267, AND 817 LINES OF FORTRAN CODE, RESPECTIVELY, TO IMPLEMENT.

FOR THE DISCUSSION THAT FOLLOWS, IT IS ASSUMED THAT THE USER IS FAMILIAR WITH THE BASIC CONCEPTS OF THE OS/360 ACCESS METHODS UNDER DISCUSSION, AS DESCRIBED IN IBM PUBLICATION (C28-6646), "IBM/360 OPERATING SYSTEM, SUPERVISOR AND DATA MANAGEMENT SERVICES".

7.1 ADDING ACCESS METHODS TO PHASE II

FOLLOWING ARE SOME BRIEF NOTES ON ADDING ACCESS METHODS TO PHASE II. IN GENERAL, THE FOLLOWING STEPS WILL BE NEEDED:

(1) MODIFY DATASET ROUTINES ("DS" MODULE) TO RECOGNIZE A NEW DATASET TYPE, AND CODE CALLS TO USER-SUPPLIED "READ", "PRINT", AND "DUMP" ACCESS-METHOD-RELATED PARAMETERS ROUTINES. IF THERE ARE NO ACCESS-METHOD-RELATED PARAMETERS, THIS STEP MAY BE OMITTED.

(2) MODIFY PROCEDURE ROUTINES ("RPR", "EXPR", AND "AUXPR" MODULES) TO RECOGNIZE NEW PROCEDURE OP CODES TO BE INTRODUCED (IF ANY), AND PROVIDE APPROPRIATE PROCEDURE STATEMENT INTERPRETATION AND EXECUTION SECTIONS.

(3) WRITE SEVERAL SUBROUTINES:

```
READ ACCESS-METHOD-RELATED PARAMETERS (IF ANY)
PRINT       "                     "          "
DUMP        "                     "          "
INTERPRET DATASET; THAT IS, SET UP THE DATASET FILES TO
  REFLECT THEIR INTERPRETATION FOR THIS DATASET ORGANIZATION.
  THIS MIGHT INCLUDE THE CREATION OF DEFAULT FILES NEEDED TO
  COMPLETELY SPECIFY THE DATASET. FOR A ONE-FILE DATASET, THIS
  ROUTINE IS NOT NECESSARY.
EXECUTION ROUTINES (CALLED BY "EXPR" OR "AUXPRE") WHICH
  SIMULATE THE EXECUTION OF THE ACCESS METHOD.
```

DETAILS ABOUT THE PRECEDING CAN BEST BE OBTAINED BY EXAMINING THE SAMPLE ACCESS METHODS WHICH HAVE BEEN PROVIDED WITH THE SYSTEM, AND WHICH COVER A RANGE OF ACCESS METHOD COMPLEXITY.

7.2 BASIC DIRECT ACCESS METHOD - TYPE "D" DATASETS

A DIRECT ACCESS DATASET REQUIRES ONLY ONE FILE TO REPRESENT IT. ANY RECORD OF THE FILE MAY BE REACHED DIRECTLY BY ADDRESS. AFTER A READ OR WRITE IS INITIATED, CONTROL IS RETURNED TO THE USER, WHO MUST ISSUE A SUBSEQUENT "WAIT" TO ENSURE THAT I/O HAS COMPLETED. "READD" AND "WRITD" ARE ALLOWABLE ACCESS OPS FOR DIRECT DATASETS.

7.3 QUEUED SEQUENTIAL ACCESS METHOD - TYPE "S" DATASETS

A SEQUENTIAL DATASET ALSO REQUIRES ONLY ONE FILE TO REPRESENT IT. A GIVEN RECORD OF THE DATASET IS ACCESSED BY PACING THROUGH THE FILE, RECORD BY RECORD, UNTIL THE REQUESTED RECORD IS REACHED (STARTING WITH RECORD ONE, IF THE FILE IS NOT ALREADY OPEN). SEQUENTIAL ACCESS ASSUMES SEQUENTIAL PROCESSING, SO RECORDS IN ADVANCE OF THE REQUESTED RECORD ARE ALSO READ IN, IN ANTICIPATION OF UPCOMING REQUESTS. CONTROL IS NOT RETURNED TO THE USER UNTIL THE RECORD REQUESTED HAS BEEN READ IN. BUFFERING, IN BOTH THE "MOVE" AND "LOCATE" MODE ARE PROVIDED. "READS", "WRITS" AND

"UPDATE" ARE ALLOWABLE A CESS OPS FOR SEQUENTIAL DATASETS, WITH UPDATE ALLOWABLE ONLY FC DATASETS BEING READ IN LOCATE MODE.

7.4 BASIC INDEX SEQUENTIAL ACCESS METHOD - TYPE "IS" DATASETS

INDEX SEQUENTIAL DATASETS ALLOW FOR RANDOM RETRIEVAL OF RECORDS ON THE BASIS OF KEY VALUE. AN "IS" DATASET ALSO ALLOWS FOR INSERTION OF NEW RECORDS WITHOUT REWRITING THE WHOLE DATASET. THESE FUNCTIONS ARE ACCOMPLISHED BY THE USE OF HIERARCHICAL INDEXES AND OVERFLOW AREAS.

TABLE SIZES RESTRICT "IS" DATASETS TO A TOTAL OF 10.

AN "IS" DATASET CONTAINS SEVERAL FILES, OF THE FOLLOWING TYPES AND DEFINITIONS ("TYPE" IS A PARAMETER OF A FILE DESCRIPTION, SEE SECTION 3.4):

PR - PRIME FILE. THE NON-OVERFLOW DATA RECORDS ARE STORED ON THIS FILE.

TI - TRACK INDEX FILE. THIS FILE SHARES EXTENTS WITH THE PRIME FILE. THAT IS, EACH PRIME CYLINDER ALSO CONTAINS ONE OR MORE TRACKS OF TRACK INDEX. FOR EACH TRACK OF PRIME FILE ON A CYLINDER, THERE ARE TWO RECORDS ON THE TRACK INDEX, ONE POINTING TO THE PRIME TRACK WHOSE HIGHEST KEY IS THE KEY OF THE RECORD, AND ONE POINTING TO THE OVERFLOW CHAIN FOR THAT TRACK, IF ANY.

OF - OVERFLOW FILE. THIS FILE CONTAINS THE OVERFLOW RECORDS FOR THE DATASET. IT MAY SHARE EXTENTS WITH THE "PR" AND "TI" FILES, IN WHICH CASE ALL THE OVERFLOW RECORDS FOR A CYLINDER ARE STORED ON THE CYLINDER ITSELF, ON TRACKS RESERVED FOR THAT USE, OR IT MAY BE CONTAINED IN AN INDEPENDENT OVERFLOW AREA. THE FORMER IS THE DEFAULT OPTION.

CI - CYLINDER INDEX FILE. THIS IS A SEPARATE FILE, CONTAINING ONE RECORD FOR EACH CYLINDER OCCUPIED BY THE PRIME FILE. EACH RECORD POINTS TO THE CYLINDER WHOSE HIGHEST KEY IS THE KEY FIELD OF THE RECORD.

MI1
MI2
MI3 MASTER INDEX FILES. "MI1" INDEXES THE CYLINDER INDEX, ONE RECORD PER TRACK OF THE "CI". EACH RECORD POINTS TO THE TRACK OF THE "CI" WHOSE HIGHEST KEY IS THE KEY FIELD OF THE RECORD. IN A SIMILAR WAY, "MI2" INDEXES "MI1", AND "MI3" INDEXES "MI2". THE CREATION OF MASTER INDEXES IS CONTROLLED BY THE "NMI" PARAMETER (SEE BELOW).

CONTENTS OF INDEX FILE RECORDS ARE DIRECT ACCESS DEVICE ADDRESS POINTERS, AND BY DEFAULT ARE ASSUMED TO BE 10 BYTES LONG. ANY OR ALL OF THE FILES OF AN "IS" DATASET MAY BE LEFT TO DEFAULT SPECIFICATION BY THE ACCESS METHOD.

FOR AN "IS" DATASET, SEVERAL ACCESS METHOD DEFINED PARAMETERS ARE NEEDED TO COMPLETELY CHARACTERIZE THE DATASET. THESE ARE CONTAINED IN A "PARAM" CARD OF THE FOLLOWING FORM (SEE SECTION 3.4):

```
PARAM    PPF,NMI,KL,CLD
```

AND ARE DEFINED AS FOLLOWS:

PARM	DEFINITION	DEFAULT
PPF	FRACTION OF PRIME AREA FULL. FOR EXAMPLE, PPF = 1.0 MEANS THAT THE PRIME AREA IS FULL, AND NO RECORDS ARE STORED IN OVERFLOW, PPF = 1.1 MEANS THAT THE PRIME AREA IS FULL, AND THERE ARE AN EQUIVALENT OF 10% OF THE PRIME RECORDS STORED IN OVERFLOW, AND PPF = .9 MEANS THAT THERE IS NO OVERFLOW, AND THE PRIME AREA IS ONLY 90% FULL (WITH THE "HOLES" DISTRIBUTED EVENLY THROUGHOUT THE PRIME AREA).	1.0
NMI	NUMBER OF TRACKS OF A MASTER INDEX TO BE ALLOWED BEFORE A HIGHER LEVEL MASTER INDEX IS CREATED	NO MI'S
KL	KEY LENGTH	10
CLD	OVERFLOW CHAIN LENGTH DISTRIBUTION. THE OVERFLOW RECORDS FOR A TRACK ARE CHAINED TOGETHER; HENCE WHEN AN OVERFLOW RECORD IS TO BE ACCESSED, ONE OR MORE RECORDS IN THE OVERFLOW AREA WILL BE READ UNTIL THE REQUESTED RECORD IS REACHED. THE USER MAY CONSTRUCT A DISTRIBUTION OF CHAIN LENGTHS (I. E., THE TOTAL NUMBER OF OVERFLOW RECORDS TO BE ASSOCIATED WITH EACH TRACK) AND PLACE ITS NAME IN THIS FIELD, OR OMIT THE FIELD AND ALLOW THE ACCESS METHOD TO CONSTRUCT THE "NATURAL DISTRIBUTION" P(N), DEFINED IN SECTION 12.2.	

THE "IS" ACCESS METHOD PROVIDES FOR RANDOM READING, WRITING, AND UPDATING OF RECORDS IN AN "IS" DATASET BY MEANS OF THE "READR", "WRITR", AND "UPDATE" ACCESS OPS, RESPECTIVELY.

A RANDOM READ IS CARRIED OUT AS FOLLOWS:

(1) READ DOWN HIGHEST LEVEL "MI" SEQUENTIALLY UNTIL THE DESIRED KEY IS "BRACKETED"

(2) CONTINUE TO READ LOWER LEVELS OF MASTER INDEX AND CYLINDER INDEX TO LOCATE CYLINDER ON WHICH THE RECORD IS LOCATED

(3) READ "TI" TO FIND TRACK ON WHICH THE RECORD EXISTS, OR IF IT

IS AN OVERFLOW RECORD, OBTAIN THE ADDRESS OF THE FIRST RECORD IN THE OVERFLOW CHAIN FOR THE TRACK

(4) READ THE PRIME RECORD, OR CHAIN OF OVERFLOW RECORDS UNTIL THE REQUIRED RECORD IS FOUND

A RANDOM WRITE (ASSUMED TO BE AN INSERT) IS CARRIED OUT AS FOLLOWS:

(1) (1)-(4) AS IN RANDOM READ

IF THE RECORD IS TO BE INSERTED IN AN OVERFLOW CHAIN:

(5) WRITE A NEW RECORD AT THE END OF THE OVERFLOW AREA, AND REWRITE THE NEXT-TO-LAST OVERFLOW RECORD READ TO UPDATE ITS CHAIN POINTER

IF THE RECORD IS TO BE INSERTED IN THE PRIME AREA:

(5) RE-WRITE THE LAST BLOCK READ, READ AND WRITE THE REMAINING BLOCKS ON THE TRACK

(6) REWRITE BOTH TRACK INDEX RECORDS FOR THIS TRACK

(7) WRITE AN OVERFLOW RECORD AT THE END OF THE OVERFLOW AREA

AN UPDATE FOLLOWING A READ MERELY REWRITES THE LAST BLOCK READ, WITH NO INDEX SEARCH REQUIRED.

8.0 I/O SUPERVISOR

THE I/O SUPERVISOR IS A PROGRAM (MODULE "AC") WHICH ACCEPTS I/O REQUESTS, MARSHALS THE REQUESTS THROUGH VARIOUS QUEUES, AND SEES THEM THROUGH TO COMPLETION. IT MAINTAINS THE CLOCK AND THE HARDWARE DEVICES (THEIR ROTATIONAL DISPLACEMENT, ACCESS ARM POSITION, AND STATUS, SUCH AS DEVICE BUSY, CHANNEL BUSY, ETC.). IT IS IMPLEMENTED AS A SIMPLE EVENT DRIVEN QUEUING MODEL, IN WHICH THE "STATIONS" ARE DEVICES, CHANNELS, AND THE CPU, AND THE "EVENTS" ARE BEGIN AND END SEEK, BEGIN AND END DATA TRANSMISSION, AND BEGIN AND END CPU PROCESSING. IT HAS ENTRY POINTS "AC", "WAIT", AND "PROC", AS DESCRIBED IN SECTION 9.3.

REQUESTS FOR SEEKS OR TRANSMITS ARE QUEUED UP ON DEVICES AND CHANNELS, RESPECTIVELY. DEVICES WHICH ARE SEEKING ARE CHAINED TOGETHER IN A DEVICE "TIME-OF-COMPLETION" (TC) CHAIN, IN ORDER OF COMPLETION. IN A SIMILAR WAY, TRANSMITTING CHANNELS ARE TIED TOGETHER IN A CHANNEL TC CHAIN.

CONTROL UNITS ARE SWITCHABLE BETWEEN CHANNELS, BUT ONCE A CONTROL UNIT IS "ATTACHED" TO A CHANNEL (BY BEING USED TO ISSUE A SEEK), IT REMAINS ATTACHED UNTIL THE REQUEST IS COMPLETED, THAT IS, UNTIL END OF DATA TRANSMISSION.

THE FOLLOWING IS A DESCRIPTION OF THE LOGIC OF THE I/O SUPERVISOR, DESCRIBING WHAT HAPPENS WHEN AN I/O REQUEST IS RECEIVED BY THE SYSTEM, AND WHAT HAPPENS WHEN THE VARIOUS TYPES OF EVENT OCCUR.

8.1 I/O REQUEST

(1) PLACE REQUEST IN REQUEST TABLE
(2) ATTACH REQUEST TO DEVICE QUEUE
(3) IF DEVICE, CONTROL UNIT, CHANNEL FREE, START SEEK

8.2 START SEEK

(1) COMPUTE DEVICE TC (END SEEK)
(2) PLACE DEVICE IN DEVICE TC CHAIN
(3) MAKE DEVICE BUSY
(4) ATTACH CONTROL UNIT TO CHANNEL, INCREMENT CU USE COUNTER

8.3 END SEEK

(1) UPDATE THE CLOCK
(2) DETACH REQUEST FROM DEVICE QUEUE
(3) ATTACH REQUEST TO CHANNEL QUEUE
(4) REMOVE DEVICE FROM DEVICE TC CHAIN
(5) IF CU AND CHANNEL ARE FREE, START TRANSMIT

8.4 START TRANSMIT

(1) COMPUTE CHANNEL TC
(2) PLACE CHANNEL IN CHANNEL TC CHAIN
(3) MAKE CHANNEL BUSY
(4) MAKE CU BUSY

8.5 END TRANSMIT (NON FORMAT WRITE)

(1) UPDATE THE CLOCK
(2) SIGNAL I/O COMPLETION TO REQUESTING PROGRAM
(3) DETACH REQUEST FROM CHANNEL QUEUE
(4) REMOVE CHANNEL FROM CHANNEL TC CHAIN
(5) FREE CHANNEL
(6) FREE DEVICE AND CU
(7) DECREMENT CU USE COUNTER. IF ZERO, DETACH CU FROM CHANNEL.
(8) REMOVE REQUEST FROM REQUEST TABLE
(9) FOR FREE CU'S ON THIS CHANNEL (BUT NOT CURRENTLY ATTACHED TO ANOTHER CHANNEL) START SEEKS ON FREE DEVICES
(10) IF THIS CHANNEL HAS A TRANSMIT WAITING, START TRANSMIT

8.6 END TRANSMIT (FORMAT WRITE) (NOT IMPLEMENTED)

(1)-(5) AS IN 8.5 (1)-(5)
(6) ATTACH REQUEST TO DEVICE QUEUE, FIRST IN LINE
(7) COMPUTE DEVICE TC FOR TRACK ERASE
(8) PLACE DEVICE IN DEVICE TC CHAIN
(9)-(10) AS IN 8.5 (9)-(10)

8.7 END TRACK ERASE (FORMAT WRITE) (NOT IMPLEMENTED)

(1) UPDATE THE CLOCK
(2) DETACH REQUEST FROM DEVICE QUEUE
(3) REMOVE DEVICE FROM DEVICE TC CHAIN
(4)-(6) AS IN 8.5 (6)-(8)
(7) IF A CHANNEL IS AVAILABLE, FOR EACH FREE DEVICE ATTACHED TO THE CU WITH PENDING SEEKS, START SEEKS
(8) IF A TRANSMIT FOR A DEVICE ON THIS CU IS WAITING ON THE CHANNEL TO WHICH THIS CU IS ATTACHED, START TRANSMIT

9.0 ORGANIZATION OF THE PHASE II SYSTEM

THIS SECTION CONSTITUTES A PRIMER ON THE IMPLEMENTATION OF THE PHASE II SYSTEM, DESCRIBING TABLES, SUBROUTINES, AND FLOW OF CONTROL IN THE SYSTEM. SECTIONS 9 & 10, TOGETHER WITH THE PROGRAM LISTINGS THEMSELVES, SHOULD PROVIDE A COMPLETE DOCUMENTATION OF THE SYSTEM.

9.1 TABLES

A BASIC PROGRAMMING DEVICE OF THE PHASE II SYSTEM IS THE "TABLE", OF WHICH THERE ARE APPROXIMATELY FOURTEEN. EACH TABLE CONTAINS INFORMATION ABOUT ALL SYSTEM ENTITIES OF A PARTICULAR TYPE. FOR EXAMPLE, THE DEVICE TABLE CARRIES DESCRIPTIONS OF EACH OF THE DEVICES WHICH A MODELER HAS SPECIFIED FOR THE SYSTEM TO BE MODELED. SECTION 3 CONTAINS DEFINITIONS OF TABLE INPUT PARAMETERS, AND SECTION 10 DEFINES INTERNAL TABLE PARAMETERS. THE PURPOSE OF THIS SECTION IS TO DESCRIBE HOW THE TABLES ARE IMPLEMENTED.

A TABLE IS IMPLEMENTED AS AN ARRAY, IN WHICH THE ROWS REPRESENT ENTITIES, AND THE COLUMNS REPRESENT ATTRIBUTES OF THE ENTITIES. HOWEVER, BY USING THE FORTRAN "EQUIVALENCE" SPECIFIER, EACH COLUMN (ATTRIBUTE) MAY BE ADDRESSED AS A ONE-DIMENSIONAL ARRAY, WITH THE SUBSCRIPT REPRESENTING THE SERIAL NUMBER OF THE ENTITY UNDER CONSIDERATION.

EACH TABLE HAS A UNIQUE TWO-CHARACTER IDENTIFIER. FOR EXAMPLE. THE IDENTIFIER OF THE DEVICE TABLE IS "DV". SIMILARLY, EACH ATTRIBUTE HAS A 1-TO-4 CHARACTER IDENTIFIER. THUS THE DEVICE "TYPE" ATTRIBUTE IS IDENTIFIED BY THE CHARACTERS "DVTYPE", AND THIS IS THE INTERNAL NAME OF THE DEVICE TYPE VECTOR. THE DEVICE TYPE OF DEVICE NUMBER 12 IS THEREFORE GIVEN BY "DVTYPE(12)".

A TABLE MAY REFERENCE AN ENTITY IN ANOTHER TABLE. FOR EXAMPLE, ONE OF THE ATTRIBUTES OF A DEVICE IS A SPECIFICATION OF THE CONTROL UNIT (IN THE CONTROL UNIT TABLE) IT IS ATTACHED TO. SUCH AN ATTRIBUTE TAKES THE FORM OF A POSITIVE INTEGER "INDEX POINTER", AND IS, IN EFFECT, A ROW NUMBER IN ANOTHER (OR THE SAME) TABLE. THUS, IF THE "CONTROL UNIT" ATTRIBUTE OF A DEVICE IS "5", THE DEVICE IS ATTACHED TO THE CONTROL UNIT WHOSE ATTRIBUTES ARE GIVEN IN ROW 5 OF THE CONTROL UNIT TABLE. IF ONE WISHED TO KNOW WHETHER OR NOT THAT CONTROL UNIT WERE BUSY, A TEST WOULD BE MADE ON "CUBUSY(5)". BY APPLYING THE APPROPRIATE LOGIC, THEREFORE, ONE CAN FIND HIS WAY AROUND THE TABLES AND EXPLORE THE RELATIONSHIPS AMONG THE ENTITIES THEREIN.

OCCASIONALLY, AN ATTRIBUTE MAY CONTAIN "REPEATING INFORMATION" ABOUT AN ENTITY. FOR EXAMPLE, AN ATTRIBUTE OF A CONTROL UNIT IS A LIST OF DEVICES ATTACHED TO IT. SUCH AN ATTRIBUTE OBVIOUSLY RE-QUIRES MORE THAN ONE STORAGE LOCATION TO SPECIFY IT. IT IS STORED IN MULTIPLE ADJACENT COLUMNS OF THE TABLE, AND A DOUBLE SUBSCRIPT CONVENTION IS USED TO ADDRESS IT. FOR EXAMPLE, THE FOURTH DEVICE ATTACHED TO CONTROL UNIT "J" WOULD BE ADDRESSED BY

"CUDV(J,4)".

ALL OF THE PROCESSING ON A GIVEN TABLE IS PERFORMED IN ONE MODULE (A SEPARATELY COMPILED PROGRAM). FOR EXAMPLE, THE MODULE "LI" HAS ROUTINES FOR READING, INTERPRETING, PRINTING, AND DUMPING THE LIST TABLES.

IN THE DISCUSSION THAT FOLLOWS, "XX" WILL BY USED AS AN ARBITRARY TABLE IDENTIFIER.

THERE ARE FOUR SCALAR PARAMETERS ASSOCIATED WITH EACH TABLE, AS FOLLOWS:

MAXXX -	THE MAXIMUM NUMBER OF ENTITIES TABLE "XX" MAY DESCRIBE
MAXAXX -	THE MAXIMUM NUMBER OF ATTRIBUTES PER ENTITY
NXX -	THE CURRENT NUMBER OF ENTITIES IN THE TABLE
NBXX -	THE NUMBER OF PRE-DEFINED (BUILT-IN) ENTITIES IN TABLE "XX"

EACH TABLE IS ACCOMPANIED BY A "TABLE MASTER", WHICH DESCRIBES THE TABLE. A MASTER HAS THREE ROWS, AND EACH COLUMN CONTAINS THREE ITEMS OF INFORMATION ABOUT AN ATTRIBUTE IN THE TABLE, NAMELY, ITS NAME, ITS TYPE, AND THE COLUMN IT OCCUPIES IN THE TABLE (THE ARRAY COLUMNS ARE NOT NECESSARILY IN THE SAME ORDER AS THE MASTER COLUMNS). THE NAME IS THE ATTRIBUTE IDENTIFIER DESCRIBED ABOVE, AND IS ALSO THE NAME USED AS A COLUMN HEADING WHEN THE TABLE IS PRINTED OUT. THE ENTITY TYPE IS ONE OF THE FOLLOWING:

I -	INTEGER
A -	ALPHAMERIC (UP TO FOUR CHARACTERS)
R -	REAL (FLOATING POINT)
D -	ALPHAMERIC (UP TO EIGHT CHARACTERS - STORED IN ADJACENT COLUMNS)
BLANK -	SECOND COLUMN OF A TYPE "D" ATTRIBUTE, OR SUCCEEDING COLUMNS OF A REPEATING ATTRIBUTE
LI -	INTEGER LIST
LR -	REAL LIST
LA -	ALPHAMERIC LIST

THE LAST THREE TYPES ARE "PSEUDO-TYPES", USED ONLY TO INDICATE THAT A LITERAL LIST IS OPTIONAL FOR THIS FIELD ON INPUT. AFTER INPUT, IT IS TREATED AS A TYPE "A" ENTITY (THE NAME OF THE LIST).

THERE ARE THREE SCALAR PARAMETERS ASSOCIATED WITH EACH MASTER TABLE, AS FOLLOWS;

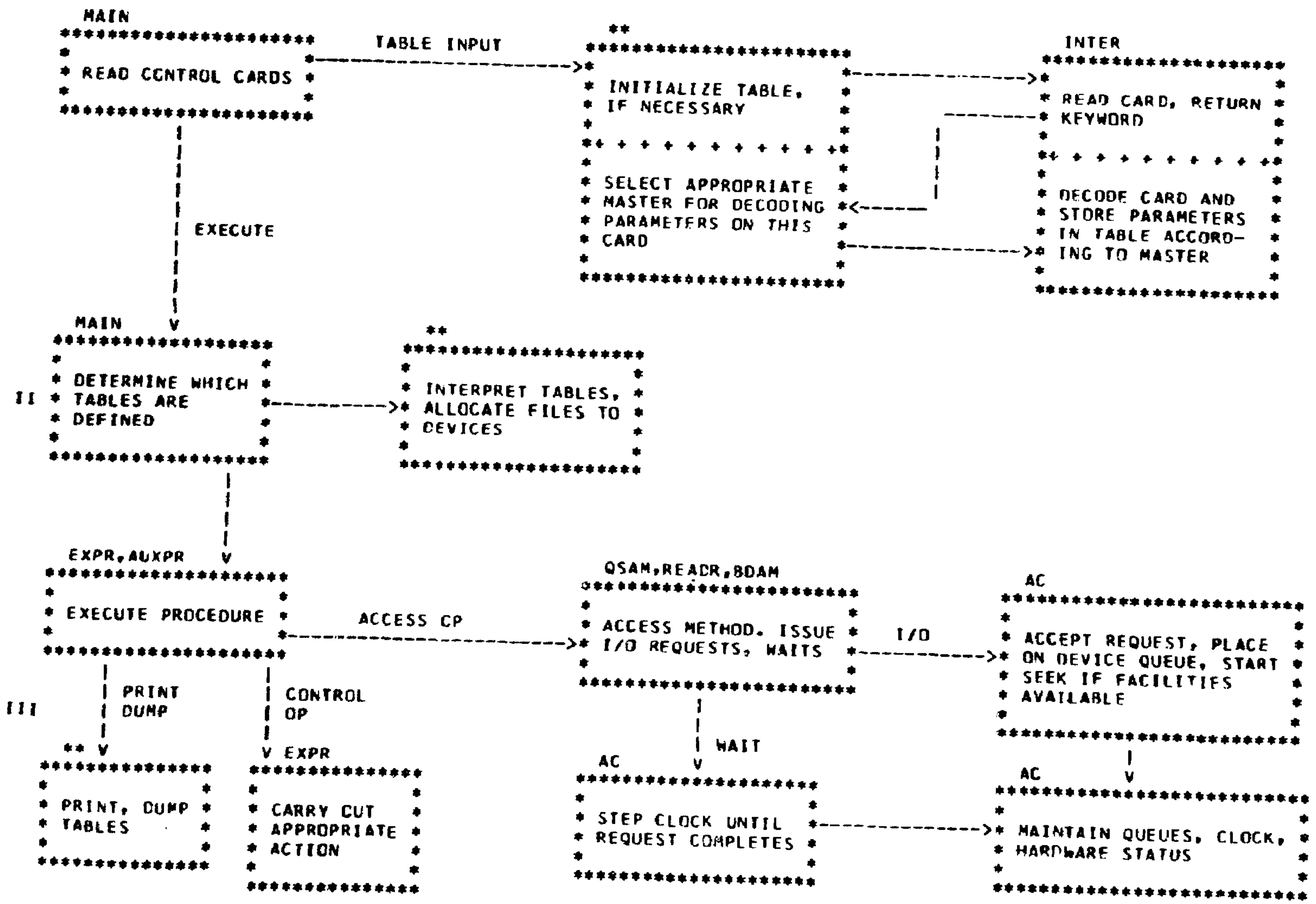
NIXX -	THE NUMBER OF INPUT PARAMETERS FOR TABLE "XX"
NPXX -	THE NUMBER OF PARAMETERS TO BE OUTPUT WHEN THE TABLE IS PRINTED
NDXX -	THE NUMBER OF PARAMETERS TO BE OUTPUT WHEN THE TABLE IS DUMPED

IN THE TABLE MASTER, INPUT PARAMETERS ARE LISTED FIRST, FOLLOWED BY ADDITIONAL PRINT PARAMETERS, FOLLOWED BY THE REMAINING PARAMETERS.

9.2 FLOW OF CONTROL

THE GROSS LOGIC OF THE PHASE II SYSTEM IS ILLUSTRATED IN FIGURE 9.2.1. EACH BOX IS LABELLED WITH THE NAME OF THE MODULE (SEE SECTION 9.3) WHICH HANDLES THE PROCESSING DESCRIBED. THERE ARE THREE MAIN PHASES TO AN EXECUTION OF THE MODEL:

PHASE I -	READ IN TABLE INPUT PARAMETERS, DOWN TO AN "*EXECUTE" CONTROL CARD
PHASE II -	INTERPRET THE TABLES; THAT IS, RESOLVE ALL INTER- AND INTRA- TABLE REFERENCES, COMPUTE INTERNAL PARAMETERS, AND PROVIDE DEFAULT SPECIFICATIONS AS NEEDED.
PHASE III-	EXECUTE THE PROCEDURE


MAIN
READ CONTROL CARDS
TABLE INPUT
**
INITIALIZE TABLE, IF NECESSARY
SELECT APPROPRIATE MASTER FOR DECODING PARAMETERS ON THIS CARD
INTER
READ CARD, RETURN KEYWORD
DECODE CARD AND STORE PARAMETERS IN TABLE ACCORDING TO MASTER
EXECUTE
MAIN
II
DETERMINE WHICH TABLES ARE DEFINED
**
INTERPRET TABLES, ALLOCATE FILES TO DEVICES
EXPR,AUXPR
EXECUTE PROCEDURE
ACCESS CP
QSAM,READR,BDAM
ACCESS METHOD. ISSUE I/O REQUESTS, WAITS
I/O
AC
ACCEPT REQUEST, PLACE ON DEVICE QUEUE, START SEEK IF FACILITIES AVAILABLE
III
PRINT DUMP
CONTROL OP
**
PRINT, DUMP TABLES
EXPR
CARRY CUT APPROPRIATE ACTION
WAIT
AC
STEP CLOCK UNTIL REQUEST COMPLETES
AC
MAINTAIN QUEUES, CLOCK, HARDWARE STATUS

** MODULE USED DEPENCS ON TABLE BEING PROCESSED

GROSS LOGIC OF THE PHASE II SYSTEM FIGURE 9.2.1

9.3 MODULES AND ENTRY POINTS

THE PHASE II SYSTEM CONSISTS OF APPROXIMATELY 25 SEPARATELY COMPILED (OR ASSEMBLED) MODULES. EACH MODULE CONTAINS ONE OR MORE ENTRY POINTS (WHICH CORRESPOND TO SUBROUTINE CALLS). THE FOLLOWING DESCRIBES THE FUNCTION OF EACH MODULE, THE TABLES IT REFERENCES (SEE SECTION 10), AND ITS ENTRY POINTS.

OF THE APPROXIMATELY 120 ENTRY POINTS IN THE SYSTEM, AROUND 60 PERFORM A STANDARD OPERATION ON A TABLE, FOR EXAMPLE, TABLE INPUT, TABLE DISPLAY, ETC. MOST OF THE TABLES HAVE FIVE ASSOCIATED ROUTINES OF THIS TYPE, AS FOLLOWS ("XX" STANDS FOR A TABLE IDENTIFIER):

RXX(NO,IP)	- READ TABLE "XX" FROM LOGICAL DEVICE "NO". PRINT THE TABLE (IN INPUT FORMAT) ON THE STANDARD OUTPUT DEVICE IF IP¬=0.
IXX	- INTERPRET TABLE "XX". INTERPRETATION IS PERFORMED FOR EACH TABLE KNOWN TO THE SYSTEM AFTER ALL TABLES HAVE BEEN INPUT, AND AN "*EXECUTE" CONTROL CARD HAS BEEN ENCOUNTERED.
PXX(NO)	- PRINT TABLE "XX" ON LOGICAL FILE "NO"
DXX(NO)	- DUMP (PRINT ALL PARAMETERS, EXTERNAL AND INTERNAL) ON FILE "NO"
FINDXX(NAME)	- THIS INTEGER FUNCTION FINDS THE ENTITY IN TABLE "XX" WHOSE NAME IS THE VALUE OF "NAME", AND RETURNS AN INDEX POINTER TO THE REQUESTED ENTITY. IT RETURNS A ZERO IF THE ENTITY IS NOT IN THE TABLE.

9.3.1 MODULES AND ENTRY POINTS

FOLLOWING IS AN ALPHABETICAL LIST OF MODULES (TOGETHER WITH THEIR ENTRY POINTS) CURRENTLY IN THE SYSTEM:

AC I/O SUPERVISOR

TABLES: BU, CH, CU, CP, DV
ENTRY POINTS:

AC(DEV,CYL,TRKP,TMT,BUFP,BUF,TYP) - INITIATE I/O REQUEST

WAIT(BUFP,BUF) - WAIT FOR A SPECIFIC REQUEST TO COMPLETE

PROC(T) - INITIATE CPU PROCESSING

RESET - RESET SYSTEM TO TIME ZERO

START - INITIALIZE SYSTEM

PIO(NO) - PRINT STATUS OF QUEUES AND HARDWARE

DBU(NO) - DUMP BUFFER TABLES

DQ(NO) - DUMP QUEUE TABLES

ALLOC FILE ROUTINES

TABLES: DP, DV, FL
ENTRY POINTS:

ALLOC - ALLOCATE SPACE FOR ALL FILES

LOCATE(FILE,REC,DEV,CYL,TRKP) - LOCATE A RECORD OF A FILE

IFL(I) - INTERPRET A FILE

CREATE(XNAME,TYPE,DEVT,IRPB,ITPC,ALLT,IALL,BTYP,NBUF,WV,CH,EXX,
        IRPC,KL,N) - CREATE A FILE

AUXPR "AUXILIARY" PROCEDURE OPS. EXTENSION TO "EXPR".

TABLES: DS, FL, LI, PR
ENTRY POINTS:

AUXPRI(NOP,I,*,*) - AUXILIARY PROCEDURE OP INTERPRET

AUXPRE(NOP,SN,*,*,*) - AUXILIARY PROCEDURE OP EXECUTE

BD BLOCK DATA FOR PRE-DEFINED SYSTEM ELEMENTS

TABLES: CH, CU, CI, CS, DP, DV, FL, LI, QU, SG, TB

```
DDAM    DIRECT ACCESS METHOD

 TABLES:  BU, FL
 ENTRY POINTS:

  READD(FIL,REC) - REAC A RECORD FROM A FILE

  WRITD(FIL,REC) - WRITE A RECORD ON A FILE

  WAITD(FIL) - WAIT FOR I/O COMPLETICN



DI      DISTRIBUTION ROUTINES

 TABLES:  DI
 ENTRY POINTS:

  RDI(NO,IP) - REAC DISTRIBUTIONS

  IDI - INTERPRET       "

  PDI(NO) - PRINT       "

  DDI(NO) - DUMP        "

  CREATD(TYPE,MODE,IPT)  - CREATE A DISTRIBUTION

  PUTD(IDIS,ARG,VAL)     - PUT AN ENTRY IN A DISTRIBUTION

  FINDDI(NAME) - FIND A DISTRIBUTION

  DIST(NO,RLO,RHI) - RETURN A RANDOM VALUE FROM A DISTRIBUTION

  IDIST(NO,LO,HI)  - SAME FOR INTEGER DISTRIBUTIONS

  DISTV(NO,ARG) - RETURN PROBABILITY OF "ARG" (DISCRETE DIST)

  DISTC(NO,ARG) - RETURN CUMULATIVE PROBABILITY OF "ARG"

  DISTA(NO,VAL) - INVERSE OF DISTC

  IDISTA(NO,VAL)  SAME FOR INTEGER DISTRIBUTIONS



DS      DATASET ROUTINES

 TABLES:  FL, DS
 ENTRY POINTS:
```

RDS(NO,IP) - READ DATASETS

IDS - INTERPRET DATASETS

PDS(NO) - PRINT DATASETS

DDS(NO) - DUMP DATASETS

FINDDS(NAME) - FIND DATASET

IDS2 - POST-ALLOCATION INTERPRET DATASETS

ERROR ERROR HANDLER

ENTRY POINTS:

ERROR(N1,N2) - SIGNAL THE SYSTEM THAT AN ERROR HAS OCCURRED

EXPR PROCEDURE INTERPRET AND EXECUTE

TABLES: DL, FL, LI, PR, QU, SG
ENTRY POINTS:

IPR - INTERPRET PROCEDURE TABLES

EXPR - EXECUTE PROCEDURE

EXPRE - RE-ENTRY POINT TO "EXPR" FOR HANDLING ERRORS

HD HARDWARE ROUTINES

TABLES: CH, CU, DP, DV
ENTRY POINTS:

RHD(NO,IP) - READ HARDWARE TABLES

PHD(NO) - PRINT HARDWARE TABLES

IDV - INTERPRET DEVICES

DDV(NO) - DUMP "

IDP - INTERPRET DEVICE PROTOTYPES

```
  DDP(NO) - DUMP      "       "

  ICU - INTERPRET CONTROL UNITS

  DCU(NC) - DUMP    "      "

  ICH - INTERPRET CHANNELS

  DCH(NC) - DUMP    "

  FINDDV(NAME) - FIND A DEVICE

  FINDCU(NAME) -  "     CONTROL UNIT

  FINDCH(NAME) -  "     CHANNEL

  FINDDP(NAME) -  "     DEVICE PROTOTYPE

  PDP(NP) - PRINT DEVICE PROTOTYPES

  RDP(NO,IP) - READ DEVICE PROTOTYPES



INTER  TABLE I/O COMMON ROUTINES

 ENTRY POINTS:

  RDCD(NC,IP,KEY) - READ AN INPUT CARD, RETURN KEYWORD

  INTERP(MASTER,CONTEN,N1,M1,N,M,IP) - INTERPRET AND STORE ONE
         CARD OF DATA ACCORDING TO MASTER AND CONTENTS TABLES
         SUPPLIED

  HEAD(MASTER,CONTEN,FORMT,BEG,END,M1,N,M,NO,N1,N2) - CONSTRUCT
         A HEADING FOR A TABLE TO BE OUTPUT AND A FORMAT STATEMENT
         FOR THE CONTENTS OUTPUT

  STORE(NAMEX,TYPEX,MAPX,MASTER,M1) - STORE NAME, TYPE, AND MASTER
         VECTORS IN MASTER

  CONVER(INPUX,OUTPUX,SPEC) - CONVERT FROM EBCDIC TO INTEGER OR
         REAL



ISAM   INDEX-SEQUENTIAL DATASET NON-EXECUTION TIME ROUTINES

 TABLES:  DP, DS, FL, IS
 ENTRY POINTS:
```

RXIS(NO,IP,NENT) - READ ISAM-RELATED DATASET PARAMETERS

PXIS(NO,NENT,I1,I2) - PRINT ISAM-RELATED PARAMETERS

DXIS(NO,NENT) - DUMP ISAM TABLE

IXIS(NO) - INTERPRET ISAM DATASET

LI   LIST ROUTINES

TABLES: LI, QU
ENTRY POINTS:

RLI(NO,IP) - READ LISTS

PLI(NO) - PRINT LISTS

DLI(NO) - DUMP LISTS

ILI - INTERPRET LISTS

FINDLI(NAME) - FIND A LIST

GET(LNO) - GET NEXT ELEMENT FROM A LIST

IGET(LNO) - SAME FOR INTEGER-VALUED LISTS

CREATL(TYPE,MQ,SIZE,LO,HS,DIS,LPT) - CREATE A LIST

PUT(LIST,ENTRY) - PUT AN ELEMENT IN A CREATED LIST

REINIT(LNO) - REINITIALIZE A LIST

PEMB(LPTX,NC1,NC2,NO) - PRINT "PROCEDURE-EMBEDDED" LIST

RLLI - RELEASE CREATED LISTS, PACK LIST TABLES

MAIN   MAIN PROGRAM - OVERALL CONTROL OF SYSTEM

OPEN   FILE OPEN AND CLOSE

TABLES: BU, FL
ENTRY POINTS:

```
  OPEN(FIL,STAT,NBUF,BTYP,CH,AV) - OPEN A FILE

  CLOSE(FIL) - CLOSE A FILE



PCP    USED BY ERROR ROUTINE (ASSEMBLY LANGUAGE)

 ENTRY POINTS:

  POP(NAME,SAVE) - RETURN NAME AND SAVE AREA OF CALLING ROUTINE.
                   REMOVE CALLING ROUTINE FROM "CALL" CHAIN.

  LINK - LINK TO ENTRY "MAIN"



QSAM   SEQUENTIAL ACCESS ROUTINES

 TABLES:  FL, BU
 ENTRY POINTS:

  READS(FIL) - READ  NEXT RECORD

  WRITS(FIL) - WRITE NEXT RECORD

  UPDATS(FIL) - UPDATE LAST RECORD READ



QU     QUALIFICATION ROUTINES

 TABLES:  DI, QU, SG
 ENTRY POINTS:

  RQU(NC,IP) - READ QUALIFICATIONS

  PQU(NC) - PRINT       "

  IQU - INTERPRET       "

  DQU(NC) - DUMP        "

  FINDQU - FIND A QUALIFICATION
```

```
RAN     RANDOM NUMBER GENERATOR (ASSEMBLY LANGUAGE)
        SEE LEWIS, GOODMAN, AND MILLER, A PSEUDO-RANDOM NUMBER
        GENERATOR FOR THE IBM 360, IBM RESEARCH REPORT RC 2330,
        JANUARY 6, 1969

 ENTRY POINTS:

  RAND(X) - RETURN A REAL RANDOM NUMBER ON (0.,1.)



RAND    RANDOM NUMBER GENERATOR

 ENTRY POINTS:

  RANDX(X,Y) - RETURN A REAL RANDOM NUMBER ON (X,Y)

  IRANDX(IX,IY) - RETURN AN INTEGER RANDOM NUMBER ON (IX,IY)



READR   INDEX-SEQUENTIAL DATASET EXECUTION-TIME ROUTINES

 TABLES:  BU, DS, FL, IS
 ENTRY POINTS:

  READR(NCF,NO) - READ A RECORD

  WRITR(NCF,NO) - WRITE A RECORD

  UPDATR(NCF) - UPDATE LAST RECORD READ



RPR     PROCEDURE TABLE ROUTINES

 TABLES:  PR
 ENTRY POINTS:

  RPR(NO,IP) - READ PROCEDURE

  DPR(NO) - DUMP PROCEDURE

  PPR(NO) - PRINT PROCEDURE

  PTI - PRINT TIMERS
```

```
SG      SEGMENT TABLE ROUTINES

 TABLES:  DS, SG
 ENTRY POINTS:

  RSG(NO,IP) - READ SEGMENTS

  ISG - INTERPRET      "

  PSG(NO) - PRINT      "

  DSG(NO) - DUMP       "

  FINDSG(NAME) - FIND A SEGMENT

  IFD - INTERPRET FIELDS

  DFD(NO) - DUMP    "

  FINDFD(NAME) - FIND A FIELD



TB      TABLE ROUTINES

 TABLES:  TB
 ENTRY POINTS:

  RTB(NO,IP) - READ TABLES

  PTB(NO) - PRINT      "

  DTB(NO) - DUMP       "

  FINDTB(NAME) - FIND A TABLE

  TABLE(TAB,ARG) - TABLE-LOOK-UP FUNCTION

  ITABLE(TAB,ARG) - SAME FOR INTEGER TABLES



TBD     TRACE BLOCK DATA FOR USE IN CONJUNCTION WITH "*TRACE"
        CONTROL CARD. THERE ARE THREE OBJECT MODULES SUPPLIED WITH
        THE SYSTEM FOR THIS PURPOSE: "TBD000" WHICH SUPPLIES NO
        TRACING (EXCEPT UNDER CONTROL OF THE "TRACE" PROCEDURE
        OP), "TBD001" WHICH TRACES ALL SUBROUTINE CALLS, AND
        "TBD002" WHICH TRACES ALL ROUTINES EXCEPT TABLE ROUTINES,
        WHICH PRODUCE VOLUMINOUS AND CONFUSING TRACE INFORMATION.
        "TBD000" IS OBTAINED BY DEFAULT, AND THE OTHERS MAY BE
        INVOKED BY INCLUSION AT LINK-EDIT TIME, AND APPROPRIATE USE
```

```
      OF THE *TRACE CONTROL CARD (SEE SECTION 2).


TIME   TASK TIMING ROUTINES (ASSEMBLY LANGUAGE)

 ENTRY POINTS:

  TIME - SET INTERVAL TIMER

  ITIME(X) - RETURN INTERVAL TIMER VALUE
```

7.3.2 ENTRY POINT - MODULE CROSS-REFERENCE

FOLLOWING IS AN ALPHABETICAL LIST OF ALL ENTRY POINTS. EACH ENTRY POINT REFERS TO THE MODULE CONTAINING IT.

ENTRY POINT	MODULE
AC	AC
ALLOC	ALLOC
AUXPRE	AUXPR
AUXPRI	AUXPR
CLOSE	OPEN
CONVER	INTER
CREATD	DI
CREATE	ALLOC
CREATL	LI
DBU	AC
DCH	HD
DCU	HD
DDI	DI
DDP	HD
DDS	DS
DDV	HD
DFD	SG
DIST	DI
DISTA	DI
DISTC	DI
DISTV	DI
DLI	LI
DPR	RPR
DQ	AC
DQU	QU
DSG	SG
DTB	TB
DXIS	ISAM
ERROR	ERROR
EXPR	EXPR
EXPRE	EXPR
FINDCH	HD
FINDCU	HD
FINDDI	DI
FINDDP	HD
FINDDS	DS
FINDDV	HD
FINDFD	SG
FINDLI	LI
FINDQU	QU
FINDSG	SG
FINDTB	TB
GET	LI
HEAD	INTER
ICH	HD
ICU	HD
ICI	DI

IDIST	DI
IDISTA	DI
IDP	HD
IDS	DS
IDS2	DS
IDV	HD
IFD	SG
IFL	ALLOC
IGET	LI
ILI	LI
INTERP	INTER
IPR	EXPR
IQU	QU
IRANDX	RAND
ISG	SG
ITABLE	TB
ITIME	TIME
IXIS	ISAM
LINK	POP
LOCATE	ALLOC
MAIN	MAIN
OPEN	OPEN
PDI	DI
PDP	HD
PDS	DS
PEMB	LI
PHD	HD
PIO	AC
PLI	LI
POP	POP
PPR	RPR
PQU	QU
PROC	AC
PSG	SG
PTB	TB
PTI	RPR
PUT	LI
PUTD	DI
PXIS	ISAM
RAND	RAN
RANDX	RAND
RDCD	INTER
RDI	DI
RDP	HD
RDS	DS
READD	BDAM
READR	READR
READS	QSAM
REINIT	LI
RESET	AC
RHD	HD
RLI	LI
RLLI	LI
RPR	RPR
RQU	QU

RSG	SG
RTB	TB
RXIS	ISAM
START	AC
STORE	INTER
TABLE	TB
TIME	TIME
UPDATR	READR
UPDATS	QSAM
WAIT	AC
WAITD	BDAM
WRITD	BDAM
WRITR	READR
WRITS	QSAM

10.0 INTERNAL PARAMETER DEFINITIONS

IN ADDITION TO ITS INPUT OR "EXTERNAL" ATTRIBUTES, AN ENTITY IN THE SYSTEM MAY ALSO HAVE ATTRIBUTES (OR PARAMETERS) WHICH ARE USED ONLY IN THE INTERNAL OPERATIONS OF THE SYSTEM. IN FACT, SOME TABLES, SUCH AS THE BUFFER (BU) TABLES, CONSIST SOLELY OF INTERNAL PARAMETERS. DEFINITIONS OF ALL INPUT PARAMETERS HAVE BEEN GIVEN IN SECTION 3, AND IT IS THE PURPOSE OF THIS SECTION TO DEFINE THE INTERNAL PARAMETERS.

FOR EACH TABLE, THE IDENTIFIER OF THE TABLE IS GIVEN (FOR EXAMPLE, "FL" FOR FILE TABLE), AND THEN THREE ITEMS OF INFORMATION ARE GIVEN FOR EACH INTERNAL ATTRIBUTE DESCRIBED BY THE TABLE:

(1) ATTRIBUTE IDENTIFIER. FOR EXAMPLE "RSIZ" FOR "RECORD SIZE".

(2) ATTRIBUTE TYPE. R = REAL, I = INTEGER, A = ALPHAMERIC

(3) ATTRIBUTE DEFINITION

AS DESCRIBED IN SECTION 9.1, A PROGRAM REFERENCE TO THE RECORD SIZE OF FILE "FIL" WOULD BE GIVEN BY "FLRSIZ(FIL)".

THOSE ATTRIBUTES CONTAINING REPEATING INFORMATION (SEE SECTION 9.1) ARE MARKED WITH AN ASTERISK (*).

SOME OF THE INTERNAL ATTRIBUTES ARE "INDEX POINTERS" (SEE SECTION 9.1) TO ENTITIES IN TABLES, AND IN SOME CASES CORRESPOND TO AN INPUT PARAMETER WHICH IS THE NAME OF THE ENTITY. IN SUCH A CASE, THE ATTRIBUTE WILL BE DEFINED AS "POINTER (XXXX)", WHERE "XXXX" IS THE NAME OF THE ATTRIBUTE.

10.1 DEVICE PROTOTYPE TABLE (DP)

PTR	I	POINTER (TB)
PTA	I	" (TABA)
PTS	I	" (TABS)

10.2 CHANNEL TABLE (CH)

TCC	I	TIME-OF-COMLETION CHAIN WORD
TC	R	TIME OF COMPLETION
CI	I	CHANNEL QUEUE CHAIN WORD: POINTER TO MOST RECENT REQUEST FOR THIS CHANNEL IN TABLE
QO	I	POINTER TO NEXT REQUEST FOR THIS CHANNEL
BUSY	I	BUSY FLAG, ¬0 IF CHANNEL IS BUSY

10.3 CONTROL UNIT TABLE (CU)

CH	I*	POINTERS TO CHANNELS IT MAY BE ATTACHED TO
DV	I*	POINTERS TO DEVICES ATTACHED TO IT
CCH	A	NAME OF CURRENT CHANNEL - MAINTAINED BY "PIC" ROUTINE
BUSY	I	BUSY FLAG, ¬0 IF CONTROL UNIT IS BUSY
USE	I	CONTROL UNIT USE COUNTER - NUMBER OF SEEKS INITIATED BY THIS CONTROL UNIT FOR WHICH TRANSMISSION HAS NOT TAKEN PLACE

10.4 DEVICE TABLE

CU	I	POINTER TO CONTROL UNIT DEVICE IS ATTACHED TO
PTR	I	POINTER (TYPE)
CYL	I	CURRENT CYLINDER UNDER HEAD
AVAL	I	NUMBER OF CYLINDERS AVAILABLE ON DEVICE
BUSY	I	BUSY FLAG, ¬0 IF DEVICE IS BUSY
TCC	I	TIME-OF-COMPLETION CHAIN WORD
TC	R	TIME OF COMPLETION
QI	I	DEVICE QUEUE CHAIN WORD: POINTER TO MOST RECENT REQUEST FOR THIS DEVICE IN TABLE
QO	I	POINTER TO NEXT REQUEST FOR THIS DEVICE

10.5 DATASET TABLE (DS)

PTR	I*	POINTERS TO FILE TABLE ENTRIES FOR FILES WHICH BELONG TO THIS DATASET
AMPT	I	POINTER TO ACCESS-METHOD-RELATED PARAMETERS ENTRY IN ACCESS METHOD PARAMETER TABLE

10.6 FILE TABLE (FL)

TNR	I	TOTAL NUMBER OF RECORDS IN FILE
RSIZ	I	RECORD SIZE
XPTR	I	POINTER TO FIRST EXTENT OF THIS FILE IN EXTENT TABLES
NEX	I	NUMBER OF EXTENTS IN THIS FILE

BPT	I	NUMBER OF BLOCKS PER TRACK OF THIS FILE
TPB	R	1/BPT
TMT	R	TRANSMIT TIME PER BLOCK
EXTP	I	POINTER (EXT)
DVTP	I	POINTER (DEVT)
BUF	I	POINTER TO BUFFER TABLE

10.7 EXTENT TABLE (EX)

PTRD	I	POINTER (DEV)
LREC	I	LAST RECORD OF FILE ON THIS EXTENT

10.8 BUFFER TABLES (BU)

MOST OF THE PARAMETERS IN THE BUFFER TABLES HAVE SIGNIFICANCE ONLY TO THE ACCESS METHOD; THAT IS, THEY CONTAIN STATUS OF AN OPEN FILE FOR USE BY THE ACCESS METHOD. TWO PARAMETERS WITH SYSTEM SIGNIFICANCE, HOWEVER, ARE:

(1) CUB - MUST BE NON-ZERO WHILE FILE IS OPEN AS AN INDICATION THAT THE BUFFER ENTRY IS IN USE

(2) BUF - THIS IS A REPEATING ATTRIBUTE, EACH INSTANCE OF WHICH CORRESPONDS TO A BUFFER AVAILABLE FOR USE BY THE FILE USING THIS BUFFER TABLE ENTRY. TWO SUBSCRIPTS ARE REQUIRED TO ACCESS THE ATTRIBUTE INFORMATION FOR A BUFFER; FOR EXAMPLE, "BUBUF(BUF,I)" WOULD CONTAIN INFORMATION RELATING TO THE "I-TH" BUFFER OF BUFFER TABLE ENTRY "BUF".

WHEN A REQUEST FOR I/O IS PRESENTED TO THE SYSTEM BY AN ACCESS METHOD (BY THE "AC" ROUTINE, SECTION 9.3), A (BUF,I) PAIR IS ALSO SPECIFIED. UPON RECEIPT OF THE REQUEST BY THE SYSTEM, A NON-ZERO VALUE IS STORED IN BUFFER "BUBUF(BUF,I)" BY THE SYSTEM, AND WHEN THE REQUEST IS SATISFIED, "BUBUF(BUF,I)" IS ZEROED OUT AGAIN.

IN THE FOLLOWING, THE PARAMETERS ARE DEFINED AS THE WERE FOR THE SEQUENTIAL ACCESS METHOD.

LSTR	I	LAST RECORD RECEIVED FROM (WRITE) OR SENT TO (READ) PROGRAM
CUB	I	CURRENT BUFFER (OF BUFFER ENTRIES) INTERFACING RECORDS TO PROGRAM

NXR	I	NEXT RECORD (OF BLOCK OR RECORDS REPRESENTEC BY "CUB") TO BE INTERFACED WITH PROGRAMS
BUF	I*	BUFFER ENTRIES
UP	I	UPDATE FLAG. =1 IF LAST RECORD READ IS TO BE UPDATED.
STAT	A	FILE STATUS. R= READ SEQUENTIAL. W=WRITE SEQUENTIAL.
LBS	I	NOT USEC
LRS	I	"
LRA	I	"
EOL	I	"

10.9 LIST TABLE (LI)

TYPN	I	LIST TYPE CODE 1 = LITERAL (LL) 2 = SEQUENTIAL (SL) 3 = RANDOM (RL,RQ) 4 = RANDOM SEQUENTIAL (RS,SQ)
MODN	I	LIST MODE CODE 1 = R 2 = I 3 = A
DPTR	I	POINTER (DIST)
PTR	I	POINTER TO FIRST ELEMENT OF THIS LIST IN LITERAL LIST ENTRY TABLE (LETAB)

THE FOLLOWING ARE DYNAMIC PARAMETERS WHICH RECORD THE CURRENT STATUS OF A LIST. AT PROCEDURE INITIATION, THE DYNAMIC PARAMETERS WILL CONTAIN THE SAME VALUES AS THEIR CORRESPONDING "STATIC", OR INPUT PARAMETERS, BUT WILL CHANGE DURING EXECUTION AS ELEMENTS ARE REMOVED FROM THE LISTS.

SIZD	I	SIZE OF LIST
ILO	I	LOW VALUE (I-LIST)
IHS	I	HIGH OR SKIP VALUE (I-LIST)
RLO	R	LOW VALUE (R-LIST)
RHS	R	HIGH OR SKIP VALUE (R-LIST)
LPTR	I	POINTER TO NEXT ELEMENT OF A LITERAL LIST

10.10 PROCEDURE TABLE (PR)

OPN	I	OPERATION CODE NUMBER
OPTR	I	POINTER (OBJ)
LPTR	I	POINTER (LIST)
SPTR	I	POINTER (SGO)
FPTR	I	POINTER (FGO)
SYNC	I	"SYNC" CHAIN WORD – TIES TOGETHER PROCEDURE STATEMENTS WHICH HAVE BEEN SYNC'ED
MPTR	I	NOT USED
NO	I	STATEMENT NUMBER (FOR PROCEDURE PRINT AND DIAGNOSTICS)
ER	I	ERROR FLAG. =1 IF PROCEDURE INTERPRETATION HAS FOUND AN ERROR IN THIS STATEMENT.
LREC	I	LAST RECORD ACCESSED BY THIS STATEMENT (IF AN ACCESS OP)

INCLUDED WITH THE PROCEDURE TABLE IS THE TIMER TABLE (TI), WITH THE FOLLOWING PARAMETERS:

NAME	A	TIMER NAME
TIMS	I	POINT IN SIMULATED TIME WHEN THIS TIMER WAS SET
TIMR	I	VALUE OF REAL TIME CLOCK WHEN THIS TIMER WAS SET. THE REAL TIME CLOCK COUNTS DOWN IN 26 MICROSECOND UNITS.

10.11 DISTRIBUTION TABLE (DI)

SIZE	I	NUMBER OF ENTRIES IN DISTRIBUTION
PTR	I	POINTER TO ENTRIES IN DISTRIBUTION CONTENTS TABLE: DEARG = TABLE OF ARGUMENTS (RANDOM VARIABLE) DEVAL = TABLE OF VALUES (CUMULATIVE DIST. VALUES)
CLAS	I	TYPE/MODE CLASS 1 = INTEGER/INTERPOLATE (OR INTEGER CONTINUOUS) 2 = INTEGER/NO INTERPOLATE 3 = ALPHAMERIC 4 = REAL/DISCRETE 5 = REAL/CONTINUOUS

10.12 QUALIFICATION TABLE (QU)

IN THE FOLLOWING, "INTERVAL" REFERS TO THE PHENOMENON INTRODUCED

BY "SORT" QUALIFICATIONS (SEE SECTION 3.5), NAMELY, THAT QUALIFICATIONS INVOLVING SORT FIELDS QUALIFY RECORDS OR SEGMENTS OVER SOME SUBSET OF THE DATASET RECORD RANGE.

VALI	I	FIELD VALUE (INTEGER)
VALR	R	FIELD VALUE (REAL)
PSQI	R	FRACTION OF SEGMENTS QUALIFYING OVER INTERVAL
PSQ	R	" " " " OVERALL
PMQI	R	FRACTION OF DATASET MASTERS QUALIFYING ON THE INTERVAL
NRQ	I	NUMBER OF DATASET MASTERS (OR RECORDS) QUALIFYING
LRQ	I	LOW RECORD QUALIFYING
HRQ	I	HIGH RECORD QUALIFYING
TYPE	I	QUALIFICATION TYPE (SEE SECTION 3.5) 1 = FIELD 2 = BOOLEAN 3 = SEGMENT
FTYP	A	FIELD TYPE (FOR TYPE 1 QUALIFICATION) I, R, A
SGPT	I	POINTER TO SEGMENT BEING QUALIFIED
FDPT	I	POINTER TO FIELD
Q13P	I	POINTER TO "Q1" OR "Q3" (SEE SECTION 3.5)
Q2PT	I	POINTER TO "Q2"
NXPT	I	POINTER TO NEXT QUALIFICATION TO INTERPRET IF THIS ONE IS MODIFIED (NOT IMPLEMENTED)
IFLG	I	INTERPRETATION FLAG 0 = QUALIFICATION NOT INTERPRETED 1 = " INTERPRETED 2 = " IN ERROR
SFLG	I	SORT FLAG ¬0 IF THIS IS A SORT QUALIFICATION

10.13 SEGMENT TABLE (SG)

SUPP	I	POINTER (SUP)
DSPT	I	POINTER (DS)
TNS	I	TOTAL NUMER OF SEGMENTS OF THIS TYPE
DSMP	I	POINTER TO THE DATASET MASTER SEGMENT OF THIS SEGMENT

NPDM	I	NUMBER OF THESE SEGMENTS PER DATASET MASTER

10.14 FIELD TABLE (FD)

SGPT	I	POINTER TO SEGMENT THIS FIELD BELONGS TO
DPTR	I	POINTER (DIST)
SIPT	I	POINTER (SIDS)

10.15 TABLE TABLES (TB)

SIZE	I	NUMBER OF ENTRIES IN TABLE
PTR	I	POINTER TO FIRST ENTRY IN TABLE CONTENTS TABLE (TBENT)

10.16 QUEUE TABLE (Q) (HOLDS REQUESTS FOR I/O)

CHAIN	I	CHAIN WORD FOR CHAINING QUEUE ELEMENTS TOGETHER
DEV	I	DEVICE REQUESTED
CYL	I	CYLINDER
TRKP	R	TRACK POSITION OF RECORD
TMT	R	TRANSMIT TIME OF RECORD
BUF	I	POINTER TO BUFFER

TYP	A	I/O REQUEST TYPE
		R = READ
		W = WRITE
		WV = WRITE VERIFY (NOT IMPLEMENTED)

11.0 SYSTEM DISTRIBUTION AND MAINTENANCE

THE PHASE II SYSTEM AS DISTRIBUTED CONSISTS OF A TAPE AND AN EXPLANATORY LISTING, AS DESCRIBED IN THE FOLLOWING.

11.1 DISTRIBUTION TAPE

THE DISTRIBUTION TAPE (800BPI, UNLABELLED) CONTAINS SIX FILES AS FOLLOWS:

(1) DATASET "SYSTEM", A PARTITIONED DATASET ("PDS") WHOSE MEMBERS ARE OBJECT MODULES OF THE SYSTEM. THE NAMES OF THESE MEMBERS ARE OF THE FORM "XXXXXNNN", WHERE "XXXXX" IS A MODULE NAME, AS GIVEN IN SECTION 9.3, AND "NNN" IS A SERIAL NUMBER, USED TO DENOTE DIFFERENT GENERATIONS OF OBJECT MODULES.

(2) DATASET "SOURCE", A PDS WHOSE MEMBERS ARE SOURCE MODULES. THE NAMES OF THE SOURCE MODULES ARE GIVEN IN SECTION 9.3.

THE SOURCE MODULES IN "SOURCE" ARE NOT COMPLETE, IN THAT THEY DO NOT CONTAIN THEIR REQUISITE TABLE SPECIFICATIONS. WHEN COMPILING SOURCE MODULES, THE REQUIRED TABLES (SEE SECTION 9.3) MUST BE (PRE-) CONCATENATED WITH THE MODULE.

(3) DATASET "LINK", A SEQUENTIAL DATASET ("SDS") USED AS THE "SYSLIN" DATASET IN THE SYSTEM LINK-EDIT. IT CONTAINS CARDS OF THE FORM "INCLUDE(XXXXXNNN)", WHERE "XXXXXNNN" IS AN OBJECT MODULE NAME.

(4) DATASET "TABLES", A PDS WHOSE MEMBERS ARE FORTRAN-LANGUAGE TABLE SPECIFICATIONS. THE NAMES OF THE MEMBERS ARE THE NAMES OF THE TABLES AS GIVEN IN SECTION 10.0.

(5) DATASET "USER", AN SDS CONTAINING THE USER GUIDE.

(6) DATASET "UPDATE", AN SDS CONTAINING UPDATES TO THE SOURCE MODULES. THEY ARE UPDATES TO THE SOURCE MODULES DISTRIBUTED WITH THE SYSTEM AND ARE ALREADY REFLECTED IN THE DISTRIBUTED OBJECT MODULES. FUTURE MINOR UPDATES, HOWEVER, CAN BE DISTRIB-UTED IN THE FORM OF UPDATE CARDS TO BE MERGED WITH THE UPDATE DECK, WHICH THEN WOULD BE RUN AGAINST THE SOURCE DECKS AND COMPILED TO OBTAIN A NEWLY UPDATED OBJECT MODULE.

11.2 DISTRIBUTION LISTING

THE DISTRIBUTION LISTING IS THE OUTPUT OF A JOB WITH THE FOLLOWING JOB STEPS:

(1) INITIALIZE DISK BY SCRATCHING DATASETS TO BE CREATED BY STEP (3), IF THEY EXIST

(2) COPY THE PHASE II MASTER SYSTEM FROM DISK TO DISTRIBUTION TAPE

```
(3) COPY THE SYSTEM DATASETS FROM TAPE BACK TO THE DISK TO TEST
(4) LINK-EDIT THE SYSTEM
(5) EXECUTE A TEST EXAMPLE
(6) DEMONSTRATE THE USE OF THE UPDATE DATASET
(7) DEMONSTRATE A COMPILE OF A SOURCE MODULE
(8) PRINT THE USER GUIDE FROM TAPE
(9) SCRATCH DATASETS CREATED BY (3)
```

12.0 MATHEMATICAL MODELS

THIS SECTION CONTAINS SOME DERIVATIONS AND FORMULAS WHOSE MOTIVATION OR FORM MAY NOT BE OBVIOUS FROM THE PROGRAM LISTINGS.

12.1 QUALIFICATION

THERE ARE SEVERAL CALCULATIONS TO BE PERFORMED IN ORDER TO DETERMINE THE RANGE AND VOLUME OF DATASET MASTER RECORDS WHICH QUALIFY FOR EACH QUALIFICATION SPECIFICATION. THESE CALCULATIONS DEPEND ON QUALIFICATION TYPE (FIELD, BOOLEAN, OR SEGMENT), AND WHETHER OR NOT SORT FIELDS ARE INVOLVED. FURTHERMORE, FOR SEGMENT QUALIFICATION, THEY DEPEND ON WHETHER OR NOT "SEG" IS SUPERIOR TO THE SEGMENT QUALIFIED BY "Q3". IN THE FOLLOWING, WE WILL DISCUSS THE FORMULAS FOR SOME OF THESE CASES; HOWEVER, WE WILL LIST ALL CASES, AND THE USER CAN TURN TO THE PROGRAM LISTINGS FOR FURTHER AMPLIFICATION.

IN THE FOLLOWING, "SEGMENT" REFERS TO THE SEGMENT BEING QUALIFIED BY THIS QUALIFICATION, AND "MASTER" REFERS TO THE DATASET MASTER SEGMENT OF "SEGMENT".

FOR EACH QUALIFICATION WE WISH TO COMPUTE:

PSQI	- FRACTION OF SEGMENTS QUALIFYING ON THE INTERVAL (IF A QUALIFICATION IS NOT A "SORT QUALIFICATION", THE INTEVAL IS THE WHOLE RANGE OF SEGMENT OCCURRENCES)
PSQ	- FRACTION OF SEGMENTS QUALIFYING OVERALL
PMQI	- FRACTION OF MASTERS QUALIFYING ON THE INTERVAL
NRQ	- NUMBER OF MASTERS QUALIFYING
LRQ	- "LOW" MASTER QUALIFYING
HRQ	- "HIGH" MASTER QUALIFYING

WE MAKE THE FOLLOWING DEFINITIONS:

N	- NUMBER OF SEGMENTS PER DATASET MASTER
ND	- TOTAL NUMBER OF DATASET MASTER SEGMENTS (= NUMBER OF RECORDS IN DATASET)

(1) A GENERAL CALCULATION

IF A FRACTION "P" OF THE SEGMENTS IN QUESTION QUALIFY, THEN (UNDER ORDINARY CIRCUMSTANCES), THE OTHER PARAMETERS ARE CALCULATED AS FOLLOWS:

```
    PSC=PSQI=P

    PMQI = 1. - PROB. THAT THE MASTER SEGMENT DOESN'T QUALIFY
         = 1.- (PROB. THAT NONE OF ITS INFERIOR SEGMENTS IN
                QUESTION QUALIFY)
         = 1.-(1. - PSQ)**N

    LRC = 1

    HRC = ND

(2) FIELD QUALIFICATION  (NAME QUAL     FLD,REL,VAL)

        LET "P" BE THE FRACTION OF FIELDS QUALIFYING.

    (A) "FLD" IS NOT A SORT FIELD - CALCULATE AS IN (1)

    (B) "FLD" IS A SORT FIELD

        LET DH = FRACTION OF FIELD VALUES "LESS THAN" THE HIGHEST
                 QUALIFYING VALUE OF THE FIELD

        PSQI = 1

        PSQ  = P

        PMQI = 1

        NRC  = PSQ*ND

        HRC  = ND*DH

        LRC  =  HRC - NRC + 1

(3) BOOLEAN QUALIFICATION  (NAME QUAL     Q1,REL,Q2)

    LET  PSQ1 =  PSQ FOR SEGMENT QUALIFIED BY Q1
         PSQ2 =   "       "         "         Q2

    (A) NEITHER Q1 OR Q2 ARE "SORT" QUALIFICATIONS

        P = PDQA*PSQB              IF REL1 = "AND"

        P = PSQA+PSQB-PSQA*PSQB  IF REL1 = "OR"

        FINISH AS IN (1)

    (B) Q1 OR Q2 IS A SORT QUALIFICATION

(4) SEGMENT QUALIFICATION  (NAME QUAL     SEG,HAS,Q3,REL2,N)

    LET:
```

```
SEG2 = THE SEGMENT QUALIFIED BY Q3
NS2  = NUMBER OF SEG2 SEGMENTS PER SEG
PS2  = PROBABILITY THAT A SEG2 QUALIFIES BY Q3
C(N1,N2) = NUMBER OF COMBINATIONS OF N1 THINGS TAKEN N2 AT
           A TIME
```

(A) SEG IS SUPERIOR TO SEG2

(A1) Q3 IS NOT A SORT QUALIFICATION

THE PROBABILITY THAT EXACTLY "M" SEG2'S QUALIFY FOR A RANDOM SEG IS GIVEN BY:

```
P= C(NS2,M)*(PS2**M)*((1-PS2)**(M-NS2))
```

FINISH AS IN (1)

(A2) Q3 IS A SORT QUALIFICATION; SEG AND SEG2 ARE ON DIFFERENT DATASETS

(A3) Q3 IS A SORT QUALIFICATION; SEG AND SEG2 ARE ON THE SAME DATASET

(B) SEG INFERIOR TO SEG2

(B1) SEG, SEG2 ON THE SAME DATASET

(B2) SEG, SEG2 ON DIFFERENT DATASETS

12.2 ISAM OVERFLOW CHAIN LENGTH DISTRIBUTION

THIS PROBLEM CAN BE STATED AS FOLLOWS:

LET: M = NUMBER OF PRIME TRACKS INITIALLY FULL

A = NUMBER OF OVERFLOW RECORDS TO BE ASSIGNED TO THE M TRACKS AT RANDOM

WHAT IS THE PROBABILITY Q(N) THAT A TRACK CHOSEN AT RANDOM HAS EXACTLY N OVERFLOW RECORDS?

THIS PROBLEM CAN BE TREATED AS A CLASSICAL OCCUPANCY PROBLEM (SEE WILLIAM FELLER, AN INTRODUCTION TO PROBABILITY THEORY AND ITS APPLICATIONS, WILEY, 1957, P. 34), WITH A SOLUTION AS FOLLOWS:

```
Q(N)=C(A,N)*((1/M)**N)*((1-1/M)**(A-N))
```

WHERE C(A,N) IS THE NUMBER OF COMBINATIONS OF A THINGS TAKEN N AT A TIME.

HOWEVER, THE DESIRED DISTRIBUTION EXCLUDES N=0, THAT IS, TRACKS WHICH HAVE NO OVERFLOW CHAIN. THE PROBABILITY P(N) THAT AN OVERFLOW CHAIN IS OF LENGTH N (N>0) IS THERFFORE:

P(N) = Q(N)/(1-P(0))

END OF PHASE II USER GUIDE

PLEASE SEND CORRECTIONS, COMMENTS, AND SUGGESTIONS TO:

P. J. OWENS
IBM RESEARCH
K06/C24
MONTEREY AND COTTLE ROADS
SAN JOSE, CALIFORNIA 95114

SECTION VIII

GLOSSARY

GLOSSARY

Definitions of some selected terminologies used in this contract report are given below for easy reference:

- Access Method - A data management program providing the retrieval, update and maintenance function for a data set.

- Access Strategy - The order and the access methods to be used in accessing all the relevant records requested in a transaction.

- Allocation - The assignment of records to specified locations of storage.

- BDAM (Basic Direct Access Method) - A specific IBM implementation of the direct access file organization.

- BISAM (Basic Indexed Sequential Access Method) - Two modes of access to ISAM data sets. In retrieval, the access method program is presented with a record identifier and it searches through indexes to retrieve the desired record. In insertion, a record is presented to the program and it is inserted in logical sequence on the basis of its identifier.

- Block (Physical Record) - A series of physically contiguous characters on a physical storage device; the unit of information transfer from peripheral devices to core memory. Blocks may contain one (unblocked) or more logical records.

- Blocking Factor - The number of logical records per (physical) block.

- BSAM (Basic Sequential Access Method) - A mode of accessing a SAM file organization where the user provides any required buffering and deblocking.

- Bucket - In the direct access organization, a set of one or more record positions associated with a particular address. Records whose identifier (or key) transforms to this address will be stored in this bucket or its extensions.

- Buffer - An area in core storage set aside to hold the contents of one block from a physical storage device.

- Byte - A generic term for a group of adjacent binary bits used as a unit. Typical examples are 8-bit byte and 6-bit byte.

- Chain - A means of interconnecting a series of information units. The connection is by means of a pointer field stored in one unit pointing to the next unit in sequence.

- Channel Program - A program to be executed by a channel.

- Cluster - A set of consecutive keys separated at both ends by a gap from other keys.

- Clockworks Model - A computerized simulation model in which complex transactions are described in terms of paths through a processing network composed of queues before specific processing stations. The lengths of the processing steps and the completion times for processing steps on a primitive transaction are determined in terms of a master clock which records simulated elapsed time.

- Collision Length - In a key to address transformation, select an arbitrary key as a starting point. The maximum number of keys following the starting key that can be mapped to distinct addresses is called the collision length.

- Control Unit - A device controlling the operation of I/O devices such as disks, tapes, pointers, etc.

- Core Memory - The section of memory attached directly to the CPU. The CPU can only directly address data and instructions stored in core memory.

- CPU (Central Processing Unit) - The computer section that provides primary interpretation of the users' programs.

- DASD (Direct Access Storage Device) - A peripheral, physical storage device, e.g., drum, disk, d[illegible] cell.

- Data Base - The totality of the collected data items in an installation.
- Data Management Program - A program in the Operating System that assists the user in accessing and managing his data files.
- Data Rate - The speed in bytes per second that a device can transmit or receive data.
- Data Set - An IBM term for a data file (and in the case of ISAM, its associated index and overflow areas).
- Debugging - The process of detecting and correcting errors in a program.
- Density of the Key Set - The ratio of the number of existing keys to the total number of possible keys in the key space.
- Domain - A mathematical concept associated with all the possible values of a set.
- DUMP - The act of printing out the entire contents of the core memory for error detection by invoking a system dump routine.
- Entity - A distinguishable object, thing or event on which information is recorded.
- Extent - A collection of stored data items that are both homogeneous and contiguous. On a disk storage device, an extent is characterized by the volume number, the starting cylinder and the number of cylinders in the extent.
- Field - The smallest information-bearing unit that may be queried and processed by a formatted file system.
- FOREM (or FSSM) (File Organization Evaluation Model (or File Structure Simulation Model)) - An off-line equation evaluation model for simulating the complex query and update transactions typical of next generation formatted file systems. (See Final Report, Contract AF 30(602)-4088 for details).

- FOREM II (File Organization Evaluation Model, Phase II) - An off-line clockworks model for simulating the complex query and update transactions typical of next generation formatted file systems.

- FORMS (File Organization Modeling System) - An on-line, combined equation evaluation-clockworks model for simulating primary key access methods. (See User's Manual complete/ under Contract AF 30602-69-C-0100 for details.)

- Full Track Blocking - One (physical) block per track.

- Galois Field - A finite field defined in the mathematical sense.

- Identifier - A group of fields which provides unique identification for a record or segment.

- Insertion - The placement of a new record into a file.

- Inverted File - A sequence of records ordered according to the magnitude of the value of a field other than the primary key field.

- I/O Supervisor - A control program for I/O operations.

- ISAM (Indexed Sequential Access Method) - A specific IBM implementation of an indexed sequential primary key file organization.

- JCL (Job Control Language) - A language for specifying the characteristics of a particular program to an IBM Operating System.

- Key-to-Address Transformation - A technique of converting a set of keys into a set of addresses on a storage device.

- Load Factor - (See Packing Factor)

- Master Segment - A segment that appears at the root of a hierarchic tree structure for a logical record. The master segment is superior to all periodic segments and appears once and only once per logical record.

- Master Index - In ISAM, any index level above the cylinder index.

- OS (Operating System) - An IBM supplied system of programs for controlling and assisting the execution of user programs.

- Overflow - Number of records that are assigned to a unit of storage space (a bucket, a track, etc.) exceed the capacity of that space.

- Pack (Volume) - A removable set of disks for a disk storage device.

- Packing Factor - In the direct access organization, the percent of possible record positions that are occupied by data records.

- Partition - Disjoint subdivision of a set of objects into smaller subunits.

- Periodic Segment - A segment that appears at a level other than the root of a hierarchic tree structure for a logical record. There will be a variable number of periodic segments per logical record.

- Processing Time - The elapsed CPU time for accessing and processing a logical record.

- QISAM (Queued Indexed Sequential Access Method) - Two modes of access to ISAM data sets. In retrieval, the access method program is presented with a record identifier and it searches through indexes to find the location of the corresponding record. It then proceeds to access, in logical sequence with automatic buffering and deblocking, as many logical records as the user requests. Updating-in-place is performed in this mode by the user requesting that the modified block in core be written back over the corresponding block on the physical storage device. The create mode is used to create and load an ISAM data organization onto physical storage.

- QSAM (Queued Sequential Access Method) - A mode of accessing a SAM file organization where the data management program provides automatic buffering and deblocking.

- Query - The process of accessing a desired set of records from a file.

- Relation - The mathematical definition of a set which is a subset of a product space $D_1 \times D_2 \ldots \times D_n$ and whose elements are of the form of ordered n-tuples $(d_1, d_2, \ldots, d_n)$.

- Repeating Segment - (See Periodic Segment)

- SAM (Sequential Access Method) - A specific IBM implementation of the sequential file organization.

- Secondary Index - A cross reference index relating the values of any non-key field to either the primary keys or the addresses of the corresponding records.

- Seek Time (Access Motion Time) - The time required to position the access mechanism at the cylinder containing the desired record.

- Segment - A specific concatenation of fields providing a description of the properties of a particular object or event.

- Trace - A time sequence recording of the occurrence of events.

- Transaction - A collection of several related processing actions in connection with an application task.

- Update - The change or modification of one or more field values in a record already in the file.

- Variable (Length) Segment - A segment which contains a variable number of characters.

- 2314 - A specific type of IBM disk device.

- 2400 - 2, 3, 4, 5, 6 - Various models of IBM tape drives.

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) IBM Research Laboratory Information Sciences Department San Jose, California 95114	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED
	2b. GROUP N/A

3. REPORT TITLE
FORMATTED FILE ORGANIZATION TECHNIQUES

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
Final Report

5. AUTHOR(S) (First name, middle initial, last name)
Michael E. Senko, et al

6. REPORT DATE June 1970	7a. TOTAL NO. OF PAGES 292	7b. NO. OF REFS 35
8a. CONTRACT OR GRANT NO. F30602-69-C-0097	9a. ORIGINATOR'S REPORT NUMBER(S)	
b. PROJECT NO. 4594		
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	
d.	RADC-TR-70-90	

10. DISTRIBUTION STATEMENT
This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Rome Air Development Center (EMIDD) Griffiss Air Force Base, New York 13440

13. ABSTRACT

This final contract report contains papers presenting several useful steps toward the creation of a more scientific discipline of formatted file design. In particular, there are papers on:

(1) The first extensive, fundamentally oriented comparison of key-to-address transforms utilizing existing formatted files.

(2) Formal, mathematical descriptions of formatted file systems that are used to provide concepts and means to deal with:

(a) the selection of indexes,
(b) direct retrieval on the basis of multiple attributed, and
(c) questions of storage and response time efficiency.

(3) The new calibration of the FOREM I Formatted File Organization Simulation Model.

(4) A new, more powerful 9000 FORTRAN statement model (FOREM II) for simulating the effects of complex file organizations, and machine configurations on efficiency and response times in a formatted file query and update environment.

DD FORM 1 NOV 65 1473

UNCLASSIFIED
Security Classification

UNCLASSIFIED
Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Computer Modeling						
Information Storage and Retrieval						
File Organization						

UNCLASSIFIED
Security Classification